ioProgrammo Anno XI - N°12 (121)

ioProgrammo

ANTEPRIMA MS ENTITY SERVER
IL FRAMEWORK TUTTO IN UNO CHE FACILITA LA GESTIONE DEI DATI IN OGNI PROCESSO DI SVILUPPO

VERSIONE PLUS RIVISTA+LIBRO+CD €9,90
VERSIONE STANDARD RIVISTA+CD €6,90

PER ESPERTI E PRINCIPIANTI

Poste Italiane S.p.A Spedizione in A.P. • D.L. 353/2003 (conv.in L.27/02/2004 n.46) art.1 comma 2 DCB ROMA Periodicità mensile • DICEMBRE 2007 • ANNO XI, N.12 (121)

MULTI SMS dal PC al telefonino

Sviluppa l'applicazione che ti consente di inviare un messaggio a più destinatari usando un comodo software lato desktop

- ✔ **PER IL COMPUTER** prepara un'interfaccia che ti permette di selezionare i destinatari e scrivere il testo
- ✔ **PER IL CELLULARE** crea un server che riceve il messaggio e l'elenco di destinazione dal PC tramite bluetooth o usb
- ✔ **TECNICA** usa le Remote API per controllare lo SmartPhone e inviare gli SMS con un solo click del mouse

SPECIALE 10 COSE DA NON FARE CON C++

Ecco le dritte degli esperti per evitare i tranelli del linguaggio "versatile" per eccellenza!

MOBILE
IL RITORNO DI PACMAN
Sviluppa un clone, per Pocket PC, del popolare gioco, utilizzando Windows Mobile e Visual Studio

.NET
MIGLIORA IL TUO STILE
Scopri il Refactoring. Rinomina oggetti e classi senza dover modificare manualmente il codice

JAVA
FATTI AIUTARE DA JMATTER
Ecco il framework che crea un'applicazione completa, e dettagliata in pochissimi secondi

PHP
PROGRAMMI PERFETTI CON LA OOP
Sotto la lente tre pattern per imparare a sfruttare tutta la potenza della programmazione ad oggetti

MULTIMEDIA
LA VALIDAZIONE DEI DATI IN FLEX
Ottieni form in "formato Flash" a prova di input utilizzando il framework di Macromedia

PATTERN
EVITIAMO GLI ERRORI DI ALIASING
La guida passo passo per implementare correttamente il Pattern Value Object

INTERNET EXPLORER ADDON

Ti spieghiamo come usare Javascript e poco altro per estendere le funzionalità del browser di casa Microsoft

PHP
HACKING DI GMAIL
Accedi a Google Mail direttamente da PHP. Ecco la libreria per utilizzare lo spazio in modo "fantasioso"

WEB
TAG CLOUD IN .NET
Crea un controllo che mostra gli argomenti più letti in un sito sfruttando la dimensione dei font

JAVA
CREA "IL" MESSENGER
Impariamo come funziona il networking in Java con un esempio divertente!

SOLUZIONI CONFRONTO FRA TESTI: SCOPRIRE LE DIFFERENZE FRA DUE O PIÙ CONTENUTI NON È UN'OPERAZIONE BANALE. ECCO COME OTTIMIZZARLA!

MULTI SMS DAL PC AL TELEFONINO • INTERNET EXPLORER ADDON • HACKING DI GMAIL • TAG CLOUD IN .NET • NETWORKING IN JAVA
EDIZIONI MASTER

Anno XI - N.ro 12 (121) - Dicembre 2007 - Periodicità Mensile
Reg. Trib. di CS al n.ro 593 del 11 Febbraio 1997
Cod. ISSN 1128-594X
E-mail: ioprogrammo@edmaster.it
http://www.edmaster.it/ioprogrammo
http://www.ioprogrammo.it

Direttore Editoriale: Massimo Sesti
Direttore Responsabile: Massimo Sesti
Responsabile Editoriale: Gianmarco Bruni
Vice Publisher: Paolo Soldan
Redazione: Fabio Farnesi
Collaboratori: R. Allegra, A. Galeazzi, F. Grimaldi, F. Smelzo, E. Bottari, O. Peli, G. Dattilo, P. Perrotta, A. Pelleriti,
Segreteria di Redazione: Rossana Scarcelli

Realizzazione grafica: Cromatika S.r.l.
Art Director: Paolo Cristiano
Responsabile grafico di progetto: Salvatore Vuono
Coordinamento tecnico: Giancarlo Sicilia
Illustrazioni: M. Veltri
Impaginazione elettronica: Francesco Cospite, Lisa Orrico, Nuccia Marra, Luigi Ferraro

Realizzazione Multimediale: SET S.r.l.
Realizzazione CD-Rom: Paolo Iacona

Pubblicità: Master Advertising s.r.l.
Via C. Correnti, 1 - 20123 Milano
Tel. 02 831212 - Fax 02 83121207
e-mail advertising@edmaster.it
Sales Director: Max Scortegagna
Segreteria Ufficio Vendite: Daisy Zonato

Editore: Edizioni Master S.p.a.
Sede di Milano: Via Ariberto, 24 - 20123 Milano
Sede di Rende: C.da Lecco, zona industriale - 87036 Rende (CS)
Presidente e Amministratore Delegato: Massimo Sesti
Direttore Generale: Massimo Rizzo

ABBONAMENTO E ARRETRATI

ITALIA: Abbonamento Annuale: IOPROGRAMMO (11 NUMERI) €59,90 SCONTO 21% SUL PREZZO DI COPERTINA DI €75,90 • IOPROGRAMMO CON LIBRO (11 NUMERI) €75,90 SCONTO 30% SUL PREZZO DI COPERTINA DI €108,90 OFFERTE VALIDE FINO AL 31/12/07.
Costo arretrati (a copia): il doppio del prezzo di copertina + € 5.32 spese (spedizione con corriere). Prima di inviare i pagamenti, verificare la disponibilità delle copie arretrate allo 02 831212.
La richiesta contenente i Vs. dati anagrafici e il nome della rivista, dovrà essere inviata via fax allo 02 83121206, oppure via posta a EDIZIONI MASTER via C. Correnti, 1 - 20123 Milano, dopo avere effettuato il pagamento, secondo le modalità di seguito elencate:

- cc/p n.16821878 o vaglia postale (inviando copia della ricevuta del versamento insieme alla richiesta);
- assegno bancario non trasferibile (da inviarsi in busta chiusa insieme alla richiesta);
- carta di credito, circuito Visa, Cartasì, o Eurocard/Mastercard (inviando la Vs. autorizzazione, il numero di carta di credito, la data di scadenza, l'intestatario della carta e il codice CVV2, cioè le ultime 3 cifre del codice numerico riportato sul retro della carta).
- bonifico bancario intestato a Edizioni Master S.p.A. c/o BCC MEDIOCRATI S.C.AR.L. c/c 0 000 000 12000 ABI 07062 CAB 80880 CIN P (inviando copia della distinta insieme alla richiesta).

SI PREGA DI UTILIZZARE IL MODULO RICHIESTA ABBONAMENTO POSTO NELLE PAGINE INTERNE DELLA RIVISTA. L'abbonamento verrà attivato sul primo numero utile, successivo alla data della richiesta.
Sostituzioni: qualora nei prodotti fossero rinvenuti difetti o imperfezioni che ne limitassero la fruizione da parte dell'utente, è prevista la sostituzione gratuita, previo invio del materiale difettato.
La sostituzione sarà effettuata se il problema sarà riscontrato e segnalato entro e non oltre 10 giorni dalla data effettiva di acquisto in edicola e nei punti vendita autorizzati, facendo fede il timbro postale di restituzione del materiale.
Inviare il CD-Rom difettoso in busta chiusa a:
Edizioni Master - Servizio Clienti - Via C. Correnti, 1 - 20123 Milano
Assistenza tecnica: ioprogrammo@edmaster.it

Servizio Abbonati:
☎ **tel. 02 831212**
@ **e-mail: servizioabbonati@edmaster.it**

Stampa: Arti Grafiche Boccia S.p.a. Via Tiberio Felice, 7 Salerno
Stampa CD-Rom: Neotek S.r.l. - C.da Imperatore - Bisignano (CS)
Distributore esclusivo per l'Italia: Parrini & C S.p.A.
Via Vitorchiano, 81 - Roma

Finito di stampare nel mese di Novembre 2007

1 Anno di Computer Bild 2006, 1 Anno di Io Programmo in DVD 2006, 1 Anno di Linux Magazine in DVD 2006, 1 Anno di Office Magazine 2006, 1 Anno di Win Magazine in DVD 2006, 360 Experience, Audio/Video/Foto Bild Italia, Auto Interactive, Calcio & Scommesse, Carlo Verdone Collection, Computer Bild Italia, Computer Games Gold, Deck, Digital Japan Magazine, Digital Music, DVD Magazine, DVD Magazine Films, Family DVD Games, Filmteca in DVD, Frank Sinatra Collection, Fred Astaire Collection, Futurama Collection, Go!OnLine Internet Magazine, Home Entertainment, Horror Mania, I Classici del Cinema Musical, I DVD di Quale Computer, I DVD di Win Magazine, I DVD de La Mia Barca, I Film di Idea Web, I Filmissimi in DVD, I Film di DVD Magazine, I Gadget de La Mia Barca, I Grandi Giochi per Pc, I Libri di Quale Computer, I Mitici all'Italiana, Idea Web, Idea Web Film, InDVD, IoProgrammo, I Tecnoplus di Win Magazine, Japan Cartoon, Jerry Lewis Collection, La mia Barca, La mia Videoteca, Linux Magazine, Miami Vice in DVD, Office Magazine, Play Generation, Play Generation Games, Play Generation Plus, Play Generation Guide e Trucchi, PC Junior, Quale Computer, Softline Software World, Sport Life, Star in DVD, Video Film Collection, Win Junior, Win Magazine Giochi, Win Magazine, Win Magazine Digital Home, Yu-Gi-Oh Collection, Le Collection.

"Rispettare l'uomo e l'ambiente in cui esso vive e lavora è una parte di tutto ciò che facciamo e di ogni decisione che prendiamo per assicurare che le nostre operazioni siano basate sul continuo miglioramento delle performance ambientali e sulla prevenzione dell'inquinamento"

Certificato UNI EN ISO 14001
N.9191.EMAS

Certificato n.6006 del 04/12/2006

ITportal
L'Universo Tecnologico
www.itportal.it

Questo mese su ioProgrammo

▼ VOTO DIECI

Ruby e Python. Se ne parla ormai da anni come due nuovi linguaggi emergenti. Ed è già paradossale che due linguaggi che esistono da moltissimi anni vengano etichettati come nuovi, ed è altrettanto paradossale che vengano etichettati come emergenti. E' ora di sdoganare questi due linguaggi come: "maturi" ed "efficaci".
C'è in Italia una sorta di diffidenza rispetto a ciò che appare come non convenzionale, per cui sia Ruby che Python nel nostro paese tendono a lavorare dietro le quinte senza mai riflettere quell'interesse che nel resto del mondo e specialmente in America è invece una realtà acquisita. Tuttavia sia Ruby che Python sono due linguaggi decisamente utili e interessanti. Primo perché hanno una curva di apprendimento vicina allo zero, secondo perché sono dotati di costrutti che li rendono particolarmente orientati alla produttività individuale e aziendale. Una parola in più voglio spenderla su Ruby On Rails, il framework per lo sviluppo Web che tanto ha suscitato clamore nel mondo e che invece qui in Italia è ancora utilizzato da una cerchia ristretta. Si tratta di un framework assolutamente "meraviglioso". Si riescono a creare complete applicazioni Web in qualcosa come due ore. E non stiamo parlando di un sitino personale, ma di applicazioni complesse aderenti al pattern MVC e con funzionalità avanzate. Provare per credere!
Da queste pagine vogliamo esortarvi a provare a programmare in Python o in Ruby non perché il linguaggio che state utilizzando adesso sia meno efficace, ma perché siamo convinti che per una quantità sterminata di applicazioni questi due linguaggi possano rendervi l'attività di programmazione molto più semplice e divertente.

All'inizio di ogni articolo, troverete un simbolo che indicherà la presenza di codice e/o software allegato, che saranno presenti sia sul CD (nella posizione di sempre *\soft\codice* e *\soft\tools*) sia sul Web, all'indirizzo http://cdrom.ioprogrammo.it.

INTERNET EXPLORER ADDON

Ti spieghiamo come usare Javascript e poco altro per estendere le funzionalità del browser di casa Microsoft

IOPROGRAMMO WEB

SISTEMA

NETWORKING

SISTEMA

DATABASE

SISTEMA

RUBRICHE

MOBILE

SISTEMA

SOLUZIONI

QUALCHE CONSIGLIO UTILE

I nostri articoli si sforzano di essere comprensibili a tutti coloro che ci seguono. Nel caso in cui abbiate difficoltà nel comprendere esattamente il senso di una spiegazione tecnica, è utile aprire il codice allegato all'articolo e seguire passo passo quanto viene spiegato tenendo d'occhio l'intero progetto.

http://forum.ioprogrammo.it

RIVISTA + LIBRO
+ CD-ROM
in edicola

I contenuti del libro

SQL SERVER ASPETTI AVANZATI

Microsoft SQL Server per molti versi non è solo un database. Lo si può considerare come un ambiente completo che espone al programmatore funzionalità complesse che abbracciano a 360 gradi la gestione dei dati. Come si possono creare programmi in T-SQL? Quali sono le modalità con cui SQL server interagisce con il .NET framework? Come posso ottimizzare al massimo le query verso il database? e ancora quali sono gli strumenti di programmazione diretta che SQL server ci mette a disposizione? A queste e a tante altre domande risponde questo Handbook. Essenziale quanto diretto, Vito Vessia ci mostra una serie di esempi concreti su come utilizzare al meglio tutte le tecniche più avanzate legate al gigantesco database server di casa Microsoft

DA T-SQL ALL'INTEGRAZIONE CON IL FRAMEWORK.NET. ECCO COME SFRUTTARE AL MASSIMO IL DATABASE DI MICROSOFT

- **Programmazione con T-SQL**
- **Integrazione con il framework .NET**
- **Catalogo e funzioni di sistema**

RIVISTA + CD-ROM in edicola

SPICES.NET SUITE 5.2.0.8

4 programmi in uno

La suite raccoglie un obfuscator, un decompiler, un modeler e un investigator.L'obfuscator consente di "criptare" il binario di un codice .NET di modo che non sia facilmente possibile disassemblarlo ed ottenerne il codice originale. Il decompiler viceversa consente di decompilare un software .NET e visionarne la struttura. A completare la suite due strumenti utilissimi, il primo: il modeler consente di progettare il proprio applicativo ed il secondo di navigarne i metadati. Si tratta una suite veramente ben studiata che fornisce al programmatore una base certa quando si vogliono integrare o modificare programmi scritti da altri.

Come usare l'interfaccia del CDRom

IL SOFTWARE
Una accurata recensione dei contenuti

IN EVIDENZA
Il top software del mese individuato dalla redazione

IL SOFTWARE
Il software diviso in categorie per una comoda consultazione

HOME
Torna alla pagina iniziale del CD-ROM

CONTATTACI
Vuoi inviare una email alla redazione con le tue richieste

TOP SITES
I siti più interessanti del mese selezionate per te

RICERCA SOFTWARE
Il database di tutti i software pubblicati da ioProgrammo anche gli arretrati

IL SOFTWARE
L'elenco del software contenuto nelle categorie

DIMENSIONE
La dimensione del software sul CD

SALVA
Clicca qui per installare o salvare il software sul tuo PC

INFO
Abbonamenti informazioni e servizi utili

News

C'È UN PO' DI GOOGLE IN MYSQL

Che a Google piacesse MySQL era arcinoto. Tant'è che spesso e volentieri sono state rilasciate patch prodotte dal gigante dei motori di ricerca tese a migliorare le caratteristiche di MySQL. Tuttavia, lentamente, la collaborazione fra MySQL AB e Google sta uscendo dai backstage per assumere una rilevanza più istituzionale. Particolare interesse rivestono le parole di David Axmark vicepresidente di MySQL AB che ha sottolineato come ormai esistano tutte le basi legali per includere ufficialmente le patch prodotte da Google nelle release ufficiali di MySQL. Le prime contribuzioni dovrebbero apparire nella versione 6.1 del DB server prevista per il 2009 e dovrebbero direzionarsi verso una migliore gestione delle repliche e del numero di istanze del database

F# ESCE DAL LABORATORIO

Da tempo si parla ormai di F#, il nuovo linguaggio di Microsoft che dovrebbe riunire un misto dei migliori linguaggi di scripting in circolazione con le tecniche ormai consolidate offerte da .NET. Fino a ieri F# è stato prima di tutto un progetto di studio prima che un oggetto concreto, oggi arriva l'annuncio della formazione di un team apposito che porti avanti lo sviluppo di F#. L'idea è di dare consistenza al linguaggio in tempi abbastanza rapidi. F# nasce per supportare campi dello sviluppo relativi alla matematica finanziaria e scientifica, per intenderci quelli che sono attualmente coperti da strumenti come Matlab

PICCOLI GRANDI BACHI PER LA JRE

Sun avvisa che è necessario aggiornare il proprio Java Runtime Environment all'ultima versione disponibile. E' stata infatti appena trovata una falla che potrebbe consentire ad un applet maligna di guadagnare privilegi che potrebbero consentirgli di danneggiare il sistema. Tutte le versioni fino alla JRE 6 Update 2 potrebbero essere affette dal bug, quindi è importantissimo aggiornare all'update 3.

DONNE NELL'IT: CHE FATICA!

Enrica Garzilli - http:/Orientalia4All.net

Un carattere indomito, uno stoicismo professionale, un senso intrepido dei legami all'azienda e una certa dose di self-promotion, sia personale che sul lavoro: queste sono le prerogative di una ventina di donne che il 19-21 ottobre si sono riunite a casa di Deirdrè Straugham, sul lago di Como. Non tutte insieme, no, ma un po' alla volta, secondo quello che ognuno aveva da fare e quando volevano, perchè tutte hanno una famiglia di cui prendersi cura, dei fidanzati che le aspettano, dei figli da mettere a letto la sera. Ma queste donne non solo solo casalinghe, mogli, madri, fidanzate o figlie ma sono ingegneri programmatrici, senior analyst, programmatrici e disegnatrici web, blogger professionale e organizzatrici di eventi, sysadmin, manager per il supporto tecnico per grandi e medie aziende, proprietarie di piccole ditte di consulenza informatica. Deirdrè, che ha ideato l'incontro, è senior web producer per la Sun e lavora come analista di contenuti per altre aziende, cioè dice loro se il messaggio che vogliono comunicare attraverso il loro sito o il loro blog viene recepito correttamente dai visitatori e possibili compratori. Abbiamo chiesto proprio a Deirdrè perchè ha invitato a casa sua tante donne professioniste dell'IT e cosa vuole ottenere. "Siamo sempre piè frustrate per la pochissima visibilità che hanno le donne nella tecnologia in Italia. Così sin da giugno qualcuna di noi ha pensato di riunirsi e ha elaborato un piano per cominciare a fare qualcosa al riguardo. Gli uomini a volte fanno una self-promotion spietata e si conoscono fra loro, si aiutano o, perlomeno, si informano l'un l'altro, a meno che non lottino per lo stesso posto. Il primo passo per noi era quindi stare insieme per conoscersi e scambiare le esperienze e per sapere esattamente cosa fare, elaborare una strategia comune. Ancora non lo sappiamo di preciso ma siamo tante in Italia. Poi, ovviamente, vogliamo costituire una lobby di donne professioniste nell'IT. In quanto tali siamo svantaggiate sia come donne che come lavoratrici, l'unica legge che, in teoria, ci tutela pienamente è il congedo per la maternità, che in Italia è molto lungo e davvero garantito. Però è del tutto legale in Italia che un possibile datore di lavoro di chieda l'età e lo stato civile e, se sei in età fertile, ottenere un lavoro a livello delle tue qualifiche o cambiare lavoro per fare carriera è difficilissimo, le aziende preferiscono sempre assumere o premiare un uomo che, secondo loro, è più presente e che < questa è la loro scusa > si concentra sul lavoro invece che sui figli." Da questa serie di incontri sono venute fuori delle proposte e un piano di azione interessante. Per prima cosa è stato aperto un gruppo di discussione su Yahoo, uno su Linkedin e un blog. Lo scopo? I gruppi di discussione sono essenzialmente per scambiarsi idee di lavoro, informazioni sui posti vacanti e le offerte di lavoro e sulle retribuzioni. Il blog è principalmente la loro "faccia pubblica" dove scambiare idee e progetti col resto del mondo in parte per aiutare altre donne e in parte per fare pubblicità e dare visibilità a loro stesse e alle loro abilità lavorative. Era stato aperto anche un gruppo dedicato su Facebook ma sembra non aver avuto molto successo ed è stato abbandonato. Il social network è molto usato dalle donne e viene usato quindi non solo per mettersi in comunicazione e per conoscersi, ma per scambiarsi informazioni utili per il lavoro o la carriera. Fra i progetti futuri c'è l'organizzazione di Girl Geek Dinner e una conferenza internazionale dove a parlare sono solo le donne. Molto importante è stata la discussione sulle qualifiche e le retribuzioni. Tutte le donne, nessuna esclusa, hanno detto che non solo le aziende nell'IT preferiscono assumere gli uomini, specie se sono programmatori, ma a parità di qualifiche, di capacità, di anzianità di lavoro e di ore dedicate all'azienda, gli uomini nell'IT sono pagati dal 10 al 20% in più delle donne. Il perchè non è chiaro, molte hanno protestato e spesso hanno avuto un avanzamento di ruolo o di retribuzione ma solo dopo aver minacciato il licenziamento.

UNA CONFERENZA "AGILE"

Enrica Garzilli

Il 23 novembre a Bologna si terrà una conferenza davvero singolare, l'Italian Agile Day. Perché singolare? Innanzi tutto perché è gratuita, poi perché è dedicata alle metodologie Agili per lo sviluppo e la gestione dei progetti software come Extreme Programming, SCRUM, Feature Driven Development, DSDM, Crystal e Lean Software Development. I partecipanti alla conferenza, che si tiene dal 2004, aderiscono, se non concretamente almeno idealmente, all'Agile Manifesto. Questo è un manifesto per lo sviluppo del software "agile" che risponde al requisito di base: la priorità di chi vi aderisce è soddisfare il cliente con continui e veloci rilasci di software. I firmatari aderiscono a quattro principi fondamentali. Il primo è che gli individui e l'interazione fra essi prevale sui processi e sui mezzi è sviluppare un programma quindi è un atto in cui la parte del leone la fanno gli hacker coinvolti e le gli utenti e non è più object-oriented, finalizzato alla pura creazione e rilascio di codice, ma un processo collaborativo; il secondo principio è che software viene considerato un processo che si sviluppa man mano e che non richiede una documentazione iniziale completa; il terzo principio è si ricerca la collaborazione del cliente, piuttosto che la negoziazione contrattuale è il cliente quindi non è un nemico con cui fare dei patti il più vantaggiosi possibile per lo sviluppatore, ma è un consumatore le cui esigenze e i cui consigli aiutano, in pratica, a sviluppare in modo più efficiente; l'ultimo principio è che il software deve rispondere ai cambiamenti, essere quindi flessibile, piuttosto che seguire un piano predeterminato. Da questo ne consegue che il rilascio del programma deve essere veloce, in modo da avere subito il feedback dal cliente, che permette a sua volta di modificarlo o di indirizzarlo nel modo giusto. Un principio base è la semplicità, perché l'arte di massimizzare il lavoro non fatto è essenziale. è un po' quello che chiunque scriva professionalmente impara a sue spese: il meno è più. Sono aperte le iscrizioni a parlare ed è stato attivato il conto PayPal per le donazioni. Per maggiori informazioni info@agileday.it o http://www.agileday.it/index.php.

WEB: CHE SIA LA "VOLTA" BUONA?

La mozione d'ordine è sempre la stessa: rendere lo sviluppo di applicazioni web immediato. Microsoft si arrovella con questo problema da tempo immemore proponendoci sistematimaticamente nuove soluzioni, eleganti, tecnologiche, sempre più semplici, eppure non manca mai di proporne una nuova, sempre migliore della precedente. La nuova soluzione targata MS si chiama "Volta" ed è stata presentata recentemente durante l'OOPSLA (Object-Oriented Programming, Languages, Systems and Applications) a Montreal. Il responsabile del progetto "Erik Meijer" così si è espresso parlando di Volta: Se pensate a quando i programmatori Visual Basic hanno fatto la loro comparsa sul mercato, vi renderete conto che a quei tempi non era facile scrivere applicazioni per Windows. Dovevate essere dei programmatori C++ e solo con l'arrivo di VB la gente è stata in grado di scrivere programmi per Windows. Adesso siamo negli anni del Web ma per certi versi siamo tornati indietro ai tempi della programmazione per Windows".

L'idea di Meijer perciò è "democratizzare" lo sviluppo di applicazioni per la rete. Nonostante queste premesse Meijer si è lanciato in una serie di discussioni sulla programmazione Multitier. Attualmente scrivere un'applicazione per il Web significa lavorare su più Tier. Una parte verrà affidata al client, ad esempio utilizzando Javascript o i CSS un'altra verrà affidata ai server e così via. Esiste secondo Meijer una stratificazione troppo massiccia delle applicazioni, così il cardine che muove "Volta" sarà quello di consentire agli sviluppatori si scrivere software trasparente a questi strati e decidere solo in seguito al deploy come debba essere distribuita l'applicazione. Come Meijer voglia raggiungere lo scopo al momento lo ignoriamo. Certo che se le premesse per semplificare lo sviluppo di applicazioni Web sono queste, vedremo quali saranno le conclusioni.

MULTI SMS DAL PC AL TELEFONINO

COLLEGHIAMO UN CELLULARE AL PC. SVILUPPIAMO UNA PICCOLA APPLICAZIONE DESKTOP CHE CONSENTE DI SCRIVERE MESSAGGI DA INVIARE A DESTINATARI MULTIPLI UTILIZZANDO ACTIVESYNC E LE REMOTE API DI WINDOWS MOBILE

SDK DI WINDOWS MOBILE

L'SDK di Windows Mobile è scaricabile all'indirizzo http://msdn2.microsoft.com/it-it/windowsmobile/bb264327.aspx

I cellulari si stanno progressivamente evolvendo fino a diventare veri e propri mini-computer con funzioni di PIM (personal information manager) e interfacce utente elaborate.
Gli Smartphone, in particolare, sono equipaggiati il sistema operativo Windows Mobile che non solo offre funzionalità evolute, ma consente a noi programmatori .NET di esplorare nuove potenzialità di sviluppo.
Il cugino "minore" di Windows supporta infatti una propria versione del .NET Framework e Visual Studio 2005 è perfettamente attrezzato per lo sviluppo di applicazioni per i dispositivi mobili. La nota dolente, per i fan di .NET, è che molte delle funzionalità esposte dalle API per Windows Mobile sono accessibili solo agli sviluppatori C++ "nativi".
Tuttavia esistono modi per aggirare tali limitazioni e continuare a sviluppare col nostro amato Framework applicazioni anche non banali.
In particolare, in questo articolo vedremo come sviluppare un'applicazione che consente di inviare un SMS a più destinatari attraverso un PC collegato ad un telefonino Smartphone attraverso una connessione ActiveSync.

PREPARARE L'AMBIENTE DI SVILUPPO

Prima di partire con il nostro programma è necessario compiere alcuni passaggi per configurare correttamente l'ambiente di sviluppo.
Innanzitutto occorre installare nella macchina l'SDK di Windows Mobile (la versione corrente dell'SDK è quella per Windows Mobile 6, ma molti dispositivi montano ancora Windows Mobile 5 quindi noi lavoreremo su quest'ultima).
Con Visual Studio è possibile anche testare le proprie applicazioni direttamente sul telefonino, tuttavia, anche se possediamo uno Smartphone, è preferibile sviluppare e debuggare il programma su un emulatore.

Per avviare un emulatore nel menu di Visual Studio andiamo alla voce ***Strumenti>Gestione emulatori dispositivo*** come in figura 1.
Scegliamo quindi uno degli emulatori di

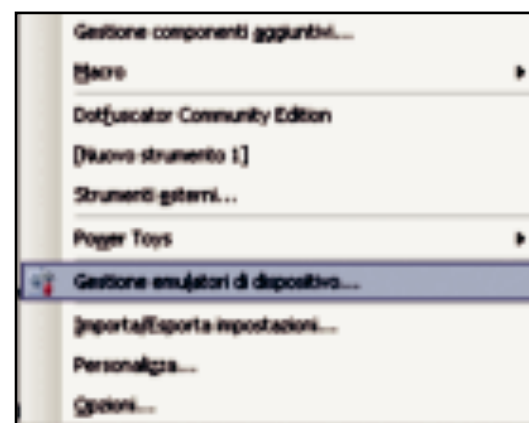



Fig. 1: ***Gestione emulatori dispositivo***

Windows Mobile nella finestra di dialogo che si aprirà (fig. 2) , attiviamo il menu contestuale con il tasto destro e clicchiamo su ***connetti***.

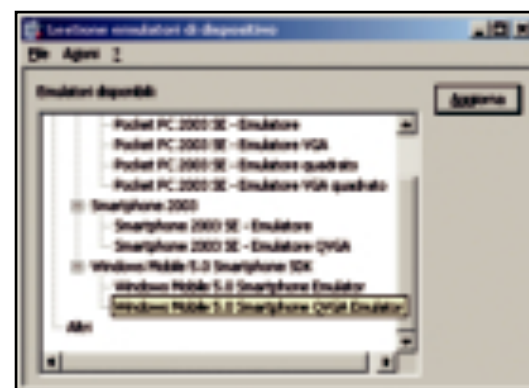



Fig. 2: ***Scelta dell'emulatore dispositivo***

A questo punto si aprirà una finestra, come quella mostrata in figura 3, che mostra l'interfaccia di emulazione.

Fig. 3: L'emulazione in funzione

Nel nostro caso però questo non basta, noi dovremo infatti sviluppare un'applicazione che utilizza una connessione ActiveSync per comunicare tra il PC ed il dispositivo, quindi è necessario anche emulare la connessione.

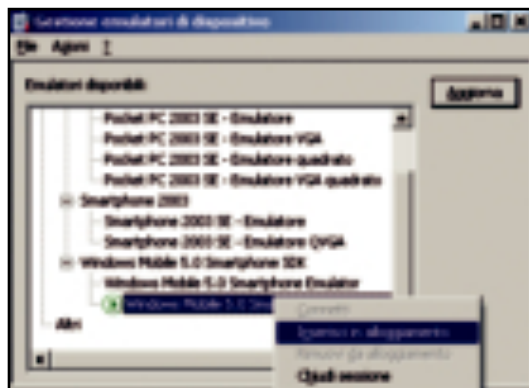



Fig. 4: avvio della connessione ActiveSync

Per far questo per prima cosa assicuriamoci di aver installato sulla macchina di sviluppo ActiveSync e che sia in esecuzione, dopodiché torniamo sulla finestra di dialogo che mostra i dispositivi di emulazione che abbiamo visto in figura 2 , attiviamo il menu contestuale sul dispositivo connesso e scegliamo la voce ***Inserisci in alloggiament***o come in **figura 4**.
A questo punto ActiveSync dovrebbe avviare la procedura guidata di riconoscimento e connessione.

IL PROGRAMMA

Terminate queste operazioni preliminari di preparazione affrontiamo il programma vero e proprio.
L'obiettivo è quello di sviluppare un'applicazione Desktop che "chiama" un processo remoto (un programma insomma) che risiede sullo Smartphone passandogli delle informazioni che rappresentano l'sms da inviare; il programma remoto compie tutte le operazioni del caso e "comunica" all'applicazione sul PC l'esito.

REMOTE API (RAPI)

Primo grosso problema : in che modo controllare lo Smartphone da un programma Desktop?
In effetti per gli sviluppatori C esiste una libreria che si chiama RAPI (Remote API) destinata proprio a questo scopo, anche in .NET si possono utilizzare le funzioni RAPI attraverso pInvoke, tuttavia rimappare una libreria di tali dimensioni non è propriamente un lavoro simpatico.
Per fortuna il sito www.opennetcf.com mette a disposizione una libreria .NET , *OpenNETCF.Desktop.Communication Library*, che fa per noi tutto il duro lavoro di colloquio con RAPI.
A corredo della libreria c'è anche un help e comunque si può consultare anche la documentazione di RAPI nell'help di Windows Mobile SDK, tuttavia vediamo in un breve esempio pratico lo schema di una chiamata di una funzione remota con RAPI che serve per ottenere informazioni sullo stato della batteria:

```
Private Sub TestRapi()
    Dim rapi As New
            OpenNETCF.Desktop.Communication.RAPI
    If rapi.DevicePresent Then
        rapi.Connect(True, 10)
        Dim pw As New
                        SYSTEM_POWER_STATUS_EX
        rapi.GetDeviceSystemPowerStatus(pw)
        Console.WriteLine(pw.BatteryLifePercent)
```

```
            rapi.Disconnect()
            rapi.Dispose()
        End If
    End Sub
```

C'è quindi prima un controllo se il dispositivo è presente (***DevicePresent***) poi la connessione (in questo caso sincrona con un timeout di 10 secondi), la chiamata di una o più funzioni RAPI ed infine la disconnessione e la distruzione dell'oggetto.

IL PROGETTO

In Visual Studio 2005 creeremo una soluzione in C# o VB.NET che ospiterà due distinti progetti : una WindowApplication che girerà sul PC e una Console Application che girerà sullo Smartphone.

Fig. 5: Progetto Console per Smartphone su Visual Studio

La base della logica di comunicazione tra i due dispositivi sarà una classe che chiameremo Message che potrà essere serializzata in XML:

1. Il PC crea un'istanza di Message dove mette numeri di telefono e testo del messaggio, la serializza su disco e la trasferisce come file allo Smartphone e avvia da remoto l'applicazione residente sul dispositivo
2. L'applicazione sullo Smartphone deserializza il file ricevuto in un oggetto Message, invia gli SMS e contrassegna ogni indirizzo con l'esito dell'invio serializzando nuovamente la classe in un file
3. A questo punto il PC recupera il file sullo Smartphone, deserializza nuovamente l'oggetto Message e legge i risultati.

La base di tutto è quindi la classe Message che vedremo adesso in dettaglio

NOTA

IMPOSTAZIONI DI SICUREZZA PER L'ESECUZIONE DI RAPI

I programmi che utilizzano RAPI, come quello descritto nell'articolo, possono non funzionare a causa delle impostazioni di sicurezza del dispositivo. In proposito è possibile consultare la sezione relativa di MSDN Library all'indirizzo http://msdn2.microsoft.com/en-us/library/ms851423.aspx

LA CLASSE MESSAGE

La classe Message ha una struttura semplice, una proprietà *Text* che rappresenta il testo del messaggio SMS e un insieme di oggetti *Destinatario* che rappresentano i destinatari del messaggio, questi oggetti sono a sua volta definiti nella classe *Destinatario* che contiene la proprietà *PhoneNumber* (numero di telefono del destinatario) e *SendResult* (risultato dell'invio) lo schema delle due classi (con i relativi attributi di serializzazione XML) è questo :

```
Imports System.Xml.Serialization
<XmlRoot("message")> _
Public Class Message
    <XmlElement("destinatario")> _
    Public Property Destinatari() As List(Of
                                        Destinatario)
...
    End Property
    <XmlElement("text")> _
    Public Property Text() As String
...
    End Property
    Friend Sub AddPhoneNumber(ByVal phoneNumber
                                        As String)
'Aggiunge un numero di telefono creando un oggetto
                                        Destinatario
' e lo inserisce in Destinatari
...
    End Sub
    Public Shared Function Deserialize(ByVal filename
                                   As String) As Message
'Deserializza un oggetto Message da un file
...
    End Function
    Public Shared Function Serialize(ByVal m As
        Message, ByVal filename As String) As Boolean
'Serializza un oggetto Message in un file
...
    End Function
End Class
Public Class Destinatario
    <XmlAttribute("phonenumber")> _
    Public Property PhoneNumber() As String
...
    End Property
    <XmlElement("sendresult")> _
    Public Property SendResult() As String
        ...
    End Property
End Class
```

COLLEGAMENTO A MESSAGE

Abbiamo visto lo schema della classe Message e abbiamo detto che essa viene usata sia dall'ap-

plicazione sullo Smartphone che da quella sul PC.
Poiché la classe deve essere identica per le due applicazioni (altrimenti le operazioni di serializzazione/deserializzazione potrebbero dare dei problemi) la cosa migliore da fare è:

- inserire la classe in uno dei due progetti della soluzione
- andare nell'altro progetto e scegliere aggiungi>elemento esistente
- scegliere il file nella cartella del primo progetto e nella finestra di dialogo scegliere aggiungi come collegamento come possiamo vedere in figura 6

Fig. 6: Aggiunta del file come collegamento

IL PROGRAMMA SULLO SMARTPHONE

Prima di vedere il funzionamento dell'applicazione su PC diamo uno sguardo a quella che andremo a creare sullo Smartphone.
L'applicazione, che è ridotta a poche righe di codice, richiede come argomento il nome del file XML che il PC avrà creato sul dispositivo; l'applicazione deserializza questo file ed invia i messaggi utilizzando la libreria *Microsoft. WindowsMobile.PocketOutlook* che avremmo aggiunto tra i riferimenti del progetto.
L'intero codice del modulo si presenta quindi così:

```
Imports Microsoft.WindowsMobile.PocketOutlook
Module MainModule
  Sub Main(ByVal args() As String)
    If args.Length < 1 Then Return
    Dim requestFilename As String = "\Program
                        Files\SmsSender\" & args(0)
    Dim responseFilename As String =
        IO.Path.ChangeExtension(requestFilename,
                                      ".response")
    Dim msg As Message =
            Message.Deserialize(requestFilename)
    IO.File.Delete(requestFilename)
```

```
    For Each d As Destinatario In msg.Destinatari
      Try
        Dim phoneNumber As String =
                              d.PhoneNumber
        Dim sms As New
          SmsMessage(phoneNumber, msg.Text)
        sms.Send()
        d.SendResult = "ok"
      Catch ex As Exception
        d.SendResult = ex.Message
      End Try
    Next
    Message.Serialize(msg, responseFilename)
  End Sub

End Module
```

IL PROGRAMMA SUL PC

Naturalmente le cose sono leggermente più complesse sul versante PC. L'applicazione sarà una Windows Application, tuttavia la finestra principale non sarà immediatamente visibile : apparirà un icona nel System Tray di Windows in modo da avere sempre la possibilità di accedere alle funzioni SMS. Per far questo la finestra sarà inizialmente non visibile, mentre verrà utilizzato (nella form stessa) un controllo NotifyIcon per gestire l'icona nel System Tray. Non ci addentreremo troppo nei particolari implementativi, ci limiteremo a dire che ci sarà naturalmente un textbox per scrivere il numero di telefono da aggiungere ad una lista di numeri, una textbox per scrivere il testo dell'SMS e due bottoni che servono rispettivamente per chiudere la finestra o per inviare il messaggio.

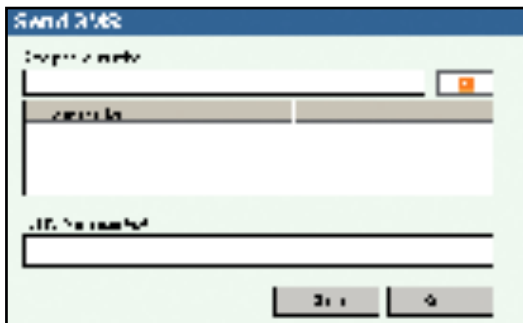



Fig. 7: La form principale del programma

COMUNICAZIONE CON IL DISPOSITIVO

Più interessante è vedere come comunicare con il dispositivo. La libreria RAPI, che abbiamo già visto, consente di effettuare alcune operazioni

NOTA

SVILUPPO SU DISPOSITIVI MOBILI

Benché Visual Studio metta a disposizione dei comodi emulatori per il test dei programmi su dispositivi mobili nella realtà il funzionamento nel mondo reale può essere a volte ben diverso.
Il consiglio è quindi di testare sempre l'applicativo anche su apparecchi reali prima di rilasciarlo.

per noi importanti:

- di invocare un processo (cioè di avviare un programma) sul dispositivo remoto consentendo anche il passaggio di parametri
- di trasferire un file dal PC al dispositivo
- di trasferire un file dal dispositivo al PC
- di cancellare un file dal dispositivo

Per comodità noi abbiamo costruito una classe, chiamata *Program*, che contiene un oggetto RAPI dichiarato in modo condiviso (*shared*) ed una serie di metodi che "wrappano" le funzioni RAPI:

```
Imports OpenNETCF.Desktop.Communication
Public Class Program
    Private Shared _rapi As RAPI
    Friend Shared ReadOnly Property rapi() As RAPI
' Istanza condivisa di RAPI
        Get
            If _rapi Is Nothing Then
                _rapi = New RAPI
            End If
            Return _rapi
        End Get
    End Property
    Private Shared Sub Connect()
        If Not rapi.Connected Then
            rapi.Connect(True, 100)
        End If
    End Sub
    Private Shared Sub Disconnect()
        If rapi.Connected Then
            rapi.Disconnect()
        End If
    End Sub
    Friend Shared Function GetDeviceProgramPath()
                                          As String
' trova la cartella "Programmi" del dispositivo
        Connect()
        Dim result As String = _
 rapi.GetDeviceSystemFolderPath(Environment.Special
                              Folder.ProgramFiles)
        Disconnect()
        Return result
    End Function
    Friend Shared Sub WriteRemoteFile(ByVal
     localFilename As String, ByVal remoteFilename As
                                         String)
' trasferisce un file sul dispositivo
        Connect()
        rapi.CopyFileToDevice(localFilename,
                              remoteFilename, True)
        Disconnect()
    End Sub
    Friend Shared Sub
     WriteRemoteFileAndDelete(ByVal localFilename As
             String, ByVal remoteFilename As String)
' trasferisce un file sul dispositivo e lo cancella dal
                                              PC
        WriteRemoteFile(localFilename,
                                 remoteFilename)
        IO.File.Delete(localFilename)
    End Sub
    Friend Shared Sub TransferFileFromDevice(ByVal
     remoteFilename As String, ByVal localFilename As
                                          String)
' trasferisce un file sul dispositivo al PC
        Connect()
        rapi.CopyFileFromDevice(localFilename,
                              remoteFilename, True)
        rapi.DeleteDeviceFile(remoteFilename)
        Disconnect()
    End Sub
    Friend Shared Function ReadRemoteFile(ByVal
               remoteFilename As String) As String
' legge file sul dispositivo copiandolo prima sul PC
        Dim result As String = ""
        Connect()
        If rapi.DeviceFileExists(remoteFilename) Then
            Dim localFilename As String =
                    IO.Path.GetFileName(remoteFilename)
            Dim tempPath As String =
Environment.GetFolderPath(Environment.SpecialFolde
                        r.LocalApplicationData)
            localFilename = IO.Path.Combine(tempPath,
                                   localFilename)
            rapi.CopyFileFromDevice(localFilename,
                              remoteFilename, True)
            result =
  My.Computer.FileSystem.ReadAllText(localFilename)
        End If
        Disconnect()
        Return result
    End Function
    Friend Shared Function
 ReadRemoteFileAndDelete(ByVal remoteFilename As
                               String) As String
' legge file sul dispositivo copiandolo prima sul PC e
                       cancellandolo dal dispositivo
        Dim result As String = ""
        If rapi.Connected Then
            result = ReadRemoteFile(remoteFilename)
            If Not String.IsNullOrEmpty(result) Then
                Connect()
                rapi.DeleteDeviceFile(remoteFilename)
                Disconnect()
            End If
        End If
        Return result
    End Function
    Friend Shared Sub ExitApp()
'Uscita dall'applicazione e distruzione di RAPI
        If rapi IsNot Nothing Then
            rapi.Dispose()
        End If
```

```
        Application.Exit()
    End Sub

    Private Shared Sub Main()
  'Entry point dall'applicazione
        Application.Run(New FrmMain)
    End Sub
End Class
```

Il processo di invio dell'SMS nel dettaglio è questo:

1. il programma PC recupera le informazioni (numeri di telefono e messaggio) e le serializza su disco locale in un file *.request* con un nome creato con una GUID.
2. il programma PC trasferisce il file *.request* sul dispositivo, lo cancella dal disco locale, avvia da remoto il programma sullo Smartphone passandogli come parametro il nome del file .request e si mette in attesa (per un numero di secondi prefissato) che nel dispositivo remoto venga creato un file *.response* avente lo stesso nome.
3. il programma sullo Smartphone deserializza in un oggetto *Message* il file *.request*, invia gli SMS ai destinatari e per ognuno di essi riporta l'esito dell'invio nella proprietà *SendResult* e infine serializza nuovamente *Message* in un file *.response*.
4. a questo punto il programma PC, che era in attesa, si accorge che nel dispositivo è presente il file *.response*, lo trasferisce sul disco e riporta i risultati dell'invio.

La cosa a prima vista può sembrare un po' macchinosa ma in realtà è l'unico modo che abbiamo per attendere che il processo sul dispositivo sia effettivamente terminato perché il metodo *CreateProcess* di RAPI non ci consente di attendere il termine dell'esecuzione del processo stesso. Tutto il codice corrispondente sarà :

```
    Private Sub Wait()
        'blocco applicazione per un secondo
        Application.UseWaitCursor = True
        Me.Cursor = Cursors.WaitCursor
        Threading.Thread.Sleep(1000)
        Application.UseWaitCursor = False
        Me.Cursor = Cursors.Default
    End Sub
    Private Function RemoteExecute(ByVal
            remoteProgramFilename As String, ByVal
    requestFilename As String, ByVal responseFilename
  As String, ByVal secondsWait As Integer) As Boolean
        'esecuzione del processo con attesa
        With Program.rapi
            .Connect()
            Try
                .CreateProcess(remoteProgramFilename,
                                          requestFilename)
            Catch ex As Exception
                MsgBox(ex.Message & vbNewLine &
                                   remoteProgramFilename)
            End Try
            For i As Integer = 1 To secondsWait
                If .DeviceFileExists(responseFilename)
                                                     Then
                    .Disconnect()
                    Return True
                Else
                    Wait()
                End If
            Next
            .Disconnect()
        End With
        Return False
    End Function
    Private Function CreateMessage() As Message
        'Creazione del Messaggio
        Dim msg As New Message
        For Each number As String In
                                     PhoneNumbers.Keys
            msg.AddPhoneNumber(number)
        Next
        msg.Text = Me.TxtSms.Text
        Return msg
    End Function
    Private Sub CreateAndSendMessage()
        If PhoneNumbers.Count = 0 Then
            MsgBox("Non ci sono numeri di telefono
  impostati!", MsgBoxStyle.Exclamation, "Errore") :
                                                   Return
        End If
        If String.IsNullOrEmpty(TxtSms.Text) Then
            MsgBox("Non hai impostato il testo del
messaggio!", MsgBoxStyle.Exclamation, "Errore") :
                                                   Return
        End If
        Dim msg As Message = Me.CreateMessage
        Dim conversationID As String =
                             Guid.NewGuid.ToString("N")
        'nome del file xml usato per la richiesta
        Dim requestFilename As String =
                              conversationID & ".request"
        'nome del file xml usato per la risposta
        Dim responseFilename As String =
                             conversationID & ".response"
        Dim localRequestFilename As String =
                   GetLocalTempFileName(requestFilename)
        Dim localResponseFilename As String =
                  GetLocalTempFileName(responseFilename)
        If Message.Serialize(msg,
                             localRequestFilename) Then
            'scrive il file nel dispositivo e lo cancella
                                              localmente
```

```
WriteRemoteFileAndDelete(localRequestFilename,
                GetRemoteFilePath(requestFilename))
        Dim RemoteResponseFilePath As String =
                GetRemoteFilePath(responseFilename)
        'esegue il processo
        Dim execResult As Boolean =
            RemoteExecute(GetRemoteProgramFilename(),
            requestFilename, RemoteResponseFilePath, 60)
        If Not execResult Then
            MsgBox("Errore Esecuzione del
        processo!", MsgBoxStyle.Exclamation, "Errore") :
                                                Return
        End If
        'trasferisce il file .response dal dispositivo al
                                                pc host
TransferFileFromDevice(RemoteResponseFilePath,
                                localResponseFilename)
        'deserializza il messaggio
        msg =
        Message.Deserialize(localResponseFilename)
        'cancella il file .response nel  pc host
        IO.File.Delete(localResponseFilename)
        'resetta la listView e l'insieme ElencoNumeri
        ElencoNumeri.Items.Clear()
        Me.PhoneNumbers = New Dictionary(Of
                                        String, String)

        'riempie la listView con i risultati
        For Each ad As Destinatario In
                                        msg.Destinatari
            Dim li As ListViewItem =
        ElencoNumeri.Items.Add(ad.PhoneNumber, 0)
            li.SubItems.Add(ad.SendResult)
        Next
    Else
        MsgBox("Errore serializzazione del
    messaggio!", MsgBoxStyle.Exclamation, "Errore") :
                                                Return
    End If
End Sub
```

ESECUZIONE DEL PROGRAMMA

Per eseguire il programma occorre per prima cosa installare il progetto per lo Smartphone sul dispositivo, ciò è possibile semplicemente cliccando con il tasto destro, in Visual Studio, sul progetto in Esplora Soluzioni, scegliendo la voce "Distribuisci" e indicando come destinazione il dispositivo attualmente connesso.
Dopo questo passaggio esplorando il File system del dispositivo con ActiveSync dovremmo trovare nel percorso "\Program Files\SmsSender" il programma *SmsSender.exe*.
A questo punto impostiamo il progetto DesktopApp (il programma per PC) come progetto di avvio e premiamo F5.
Nella System Tray scegliamo *Send SMS* ed impostiamo i numeri di telefono ed il messaggio (Figura 8)

Fig. 8: Impostazione dei numeri di telefono ed del messaggio

Clicchiamo su Send e, dopo qualche secondo, dovremo vedere nella ListView il messaggio di conferma accanto ai numeri di telefono (Figura

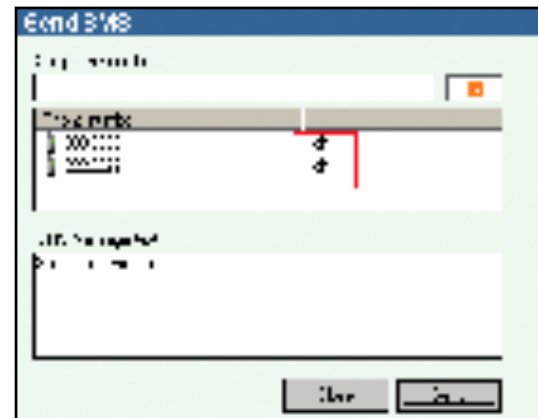



Fig. 9: Conferma dell'invio

CONCLUSIONI

In questo articolo abbiamo visto come controllare da remoto il dispositivo mobile, in questo caso per permettere l'invio di SMS da un PC collegato allo Smartphone. Il codice di esempio non è stilisticamente perfetto (alcune problematiche potevano essere affrontate anche in maniera più elegante, ma ciò avrebbe comportato una minore comprensibilità del codice) e mancano alcune funzioni lato desktop (ad esempio selezione da indirizzi di Outlook, possibilità di salvare e riutilizzare le liste di contatti ecc...) comunque può essere considerato come una base per prendere confidenza con la tecnica descritta. Al lettore il lavoro di adattamento e perfezionamento!

Francesco Smelzo

NOTA

Francesco Smelzo è specializzato nello sviluppo in ambiente Windows con particolare riferimento ad applicazioni in ambiente .NET sia web-oriented che desktop. Il suo sito web è www.smelzo.it. Come sempre è a disposizione per ricevere suggerimenti o richieste sull'articolo all'indirizzo di posta elettronica francesco@smelzo.it

PHP INCONTRA GOOGLE MAIL

HAI MAI PENSATO DI USARE GMAIL COME ARCHIVIO FTP? TI PIACEREBBE ACCEDERE ALLA POSTA DIRETTAMENTE DA UNA TUA WEB APPLICATION PHP? TI PRESENTIAMO LA LIBRERIA CHE TI CONSENTE DI FARE TUTTO CIÒ. POCHE RIGHE DI CODICE E GMAIL NON AVRÀ PIÙ SEGRETI!

Tra le tante librerie di cui Php dispone per l'interazione con diverse tipologie di software, in quest'articolo concentreremo l'attenzione su una in particolare: *Libgmailer*, collezione di funzioni scritta per integrae PHP con Gmail. Vedremo come connetterci, come fare il login alla casella di posta elettronica, creare l'oggetto Gmailer , prelevare e-mail inviate, inviare noi stessi delle e-mail. Dopo aver spiegato il funzionamento, sulla base delle conoscenze acquisite, svilupperemo una piccola applicazione.

DESCRIZIONE DELLA LIBRERIA

LibGmailer consente di connettersi al sito di Gmail e ricercare le informazioni richieste dall'utente, effettuando, in maniera automatica, le operazioni di login, gestendo tutte le operazioni possibili con un servizio di webmail, compresa la selezione di messaggi da prelevare in base a determinati attributi, come per esempio la suddivise in categorie o la selezione di messaggi flaggati (con *star* o *label*) in modo da poter essere contraddistinti. In maniera più analitica, *Gmailer*, la classe su cui lavoreremo (la quale è una classe funzionale e dunque ha bisogno che le sue funzioni siano integrate in un'applicazione completa per poter ottenere un risultato) si preoccupa di scrivere, comporre, salvare bozze, inviare, applicare etichette alle e-mail, ma anche di rimuoverle, applicare la stella di Gmail, rimuoverla, caricare e scaricare attachment: tutto questo per quel che riguarda la gestione delle e-mail. La prima cosa che *Gmailer* fa per noi è quella di connettersi a *Gmail*: per portare a termine questo processo e "armeggiare" con il traffico *http/https*, essa utilizza l'estensione *curl* di *php*, senza cui non funzionerebbe.

CHE COS'È CURL?

Curl sta per *Client-Url* e serve per il trasferimento di file attraverso la sintassi *Url*, che supporta i protocolli *Ftp*, *Ftps*, *Http*, *Https*, *Tftp*, *Telnet* e altri ed è utilizzato quando bisogna andare a ritrovare contenuti all'interno di siti web e prelevare o, comunque, scambiare dei file. Affinché esso sia utilizzato in ambiente Php, è necessario attivarne l'estensione. Le operazioni per fare questo sono davvero molto semplici. Innanzitutto bisogna che il pacchetto Curl sia installato e, affinché funzioni su Php, esso deve essere almeno nella versione Curl 7.0.2-beta. Se stiamo usando una versione di Php successiva alla 4.2.3, bisogna che la versione Curl sia la 7.9.8. Per poter utilizzare tale supporto, occorre compilare Php con l'opzione

```
--with-curl[=DIR],
```

dove *DIR* sta per la directory che contiene le directory di tipo *lib* e *include*. Se stiamo usando una versione di Php successiva alla 4.3.0, invece, possiamo configurare Ph all'uso di curl per gli Url stream

```
--with-curlwrappers
```

Se siamo utenti Windows, dobbiamo abilitare il tutto copiando *libeay32.dll* e *ssleay32.dll* dalla directory *PHP/Win32* alla cartella di Windows *C:\\Windows\system32* o *C:\\Winnt\system*.

L'UTILIZZO DI LIBGMAILER

Come utilizzare allora la libreria? Come si interagisce con essa? Costruiamo insieme il codice necessario a sfruttare le sue funzionalità e a renderla operativa. La prima operazione che eseguiremo sarà quella di prelevare un'e-mail, che sceglieremo in base ad un'etichetta definita dall'utente.
Proviamo per prima cosa a prelevare un'e-mail che abbiamo etichettato come *Label*: questa semplice applicazione ci permetterà di approfondire come richiamare la classe *GMailer*, come effettuare la connessione, prelevare i dati di login per Gmail e fare i controlli per individuare se ci sono problemi di

installazione o problemi con la connessione.

```
<?php
require("include/libgmailer.php");
require("include/config.php");
```

Oltre al tag che contraddistingue l'inizio di porzione di codice *php*, provvediamo al richiamo dei file che ci servono: la libreria *libgmailer.php* e un file *config.php*, nella cartella include *Libgmailer* si trova facilmente in rete (ad esempio http://sourceforge.net/project/showfiles.php?group_id=116304 dove è presente la versione 0.9.0 beta 8). A cosa serve config.php? è un file di configurazione che permette di memorizzare i nostri dati per l'autenticazione come utente di Gmail. Nel box ad esso dedicato c'è il semplice codice che lo compone.

```
$gm = new GMailer();
  if (!$gm->created) {
```

Creiamo ora l'oggetto GMailer:

```
  die("Non è stato possibile creare l'oggetto Gmailer:
                                ".$gm->lastActionStatus());
              } else {
```

il costruttore, così come è stato strutturato, farà un controllo per verificare se l'estensione Curl è disponibile o meno. Da ciò l'importanza della sua abilitazione come sottolineato nelle premesse. Poiché non abbiamo un dato di output che ci riveli l'avvenuta creazione, dobbiamo accertarci di questo con il metodo *GMailer::$created*. Nel caso di fallimento, mandiamo un messaggio di errore usando il metodo *lastActionStatus()* che descrive, con una stringa, il problema (nello specifico, nel caso in cui si fosse verificato il problema il messaggio sarebbe stato "*no curl*") .

```
$gm->setLoginInfo($user, $password, $timezone);
```

Se l'operazione è andata a buon fine , possiamo acquisire le informazioni di login, settate nel file *config.php* . Passiamo ora al prelievo vero e proprio!

```
if ($gm->connect()) {
          $gm->fetchBox(GM_LABEL, "label", 0);
          $snapshot = $gm->getSnapshot(GM_LABEL);
          if ($snapshot)
                print( "Numero totale  delle conversazioni
                        con la mia etichetta =  " .
                                      $snapshot->box_total);
          } else {
                die("La connessione  è fallita perché
                        ".$gm->lastActionStatus());
          }
}
```

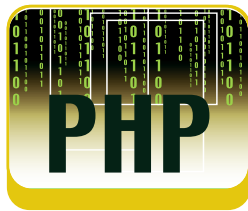
La prima operazione è quella di connettersi, su questa porazione facciamo subito un controllino per vedere se la connessione è andata a buon fine se fallisce salta alla fine stampando il messaggio di errore ricevuto in output da *lastActionStatus()*. Se tutto va bene, invece, utilizza il metodo *GMailer::$fetchBox* per prelevare il risultato di una ricerca il cui argomento è di tipo *GM_LABEL* e che viene settato nel secondo parametro del metodo. Il terzo parametro di input, invece, rappresenta dove il cursore si deve posizionare per effettuare il prelievo: visto che a noi interessava tutto il contenuto, lo abbiamo settato a *0*. A questo punto possiamo richiamare il metodo *GMailer::getSnapshot* che si preoccupa di fornire un oggetto di tipo *snapshot*, creato appositamente per strutturare i risultati della query creata avendo come argomento *GM_LABEL*. Se questo è stato fatto, contiamo infine il numero degli *snapshot* trovati con l'etichetta e mandiamoli in output.

VARI TIPI DI FETCH

Approfondiamo ora il discorso.. Il codice di prima ci ha permesso di prendere un tantino dimestichezza con la libreria introducendo il discorso della *fetch*. Creare un'applicazione per fare una *fetch* solo sulle *label* può essere limitativo, pertanto ci cimenteremo subito in qualcosa di più completo. Il metodo *fetchbox()* permette, di volta in volta, grazie all'uso di diverse costanti, di impostare i parametri che serviranno a diversificare i tipi di prelievo: *GM_STANDARD* per prelevare tutti gli standard box (*Inbox*, *Sent*, *All*, *Starred*, *Spam* e *Trash*), *GM_LABEL* per prelevare le e-mail che vengono identificate da una determinata etichetta, *GM_QUERY* per impostare una query, etc.. Il metodo infatti è del tipo: *GMailer::$fetchBox(par1, par2, par3*), dove par1 è la costante che identifica il tipo a cui appartiene par2. Del terzo abbiamo già detto.

```
require("include/libgmailer.php");
require("include/config.php");
$gm = new GMailer();
  if (!$gm->created) {
              die("Non è stato possibile creare l'oggetto
                    Gmailer: ".$gm->lastActionStatus());
              } else {
                $gm->setLoginInfo($user, $password,
                                              $timezone);
                if ($gm->connect()) {
```

Quella precedente è la prima parte che abbiamo già descritto, per cui possiamo passare subito alla parte nuova.

```
    switch ($Che_operazione_vuoi_fare) {
```

```
case 1:
    $gm->fetchBox(GM_STANDARD, "inbox", 0);
    $snapshot = $gm->getSnapshot(GM_STANDARD);
break;
```

Il codice prevede la possibilità di scegliere quale operazione effettuare mediante uno switch.
Il caso 1 usa la costante *GM_STANDARD* che viene impostata ad inbox: questo vuol dire che chiediamo di prelevare tutto il contenuto della Inbox dell'account.

```
case 2:
    $gm->fetchBox(GM_LABEL, "my_label", 0);
    $snapshot = $gm->getSnapshot(GM_LABEL);
    break;
```

Il *case 2* si occupa invece di ricercare tutte le e-mail che sono etichettate come my_label, il cui tipo è definito dalla costante *GM_LABEL*.

```
case 3:
    $gm->fetchBox(GM_QUERY, "From: friend
                                @domain.it", 0);
    $snapshot = $gm->getSnapshot(GM_QUERY);
    break;
```

Analogamente, il *case 3*, il cui compito è processare la query tra virgolette e ricercare tutte le e-mail che sono state mandate da quell'indirizzo.

```
case 4:
    $gm->fetchBox(GM_CONTACT, "all", 0);
    $snapshot = $gm->getSnapshot(GM_CONTACT);
break;
```

Infine, agiamo sulla rubrica: il risultato di questa *fetch* è uno *snapshot* contenente i dati delle persone che abbiamo nella nostra rubrica Gmail.

```
default:
        die();
  }
}
```

Infine, se nella nostra interfaccia grafica che interagisce con il codice *php* non è stata fatta nessuna scelta che sia contemplata nello *switch*, il processo viene interrotto e si esce dal programma. Le righe che seguono sono di conclusione e sono stata già commentate nel precedente paragrafo.

```
                    else {
            print("La connessione è fallita
                perché ".$gm->lastActionStatus());
                                    }
        }
```

MANDIAMO UN'EMAIL

Utilizziamo ora il metodo *GMailer::$send* per inviare un'email dall'account Gmail.
Il metodo restituisce un valore booleano per controllare l'avvenuto invio della mail, che può essere utilizzato insieme a *lastActionStatus()*, come abbiamo già fatto precedentemente. *Send()* accetta come parametri di input tutti i normali componenti di un'email: a chi inviare (*A:*, *Cc:*, *Ccn:*), il soggetto del messaggio, il messaggio stesso, allegati, e via di seguito. Descriviamo un esempio di implementazione che ne faccia uso.
Dopo aver definito gli *include*, creato la *Gmailer* e impostata la connessione (il cui codice è analogo alle parti precedenti) , modifichiamo il codice a partire dal punto (1).

```
$to = "a_chi_inviare
                @dominio.com, amico@azienda.com";
$subj = "Invio di un'email con attachment";
$message = "E' arrivato?";
$cc = "a_chi_inviare2 @dominio.it";
$attachments = array("path/file.ext ", "path/file2.ext ");
```

Abbiamo definito le variabili che useremo per l'imput del metodo: notiamo come siano passate tutte quante come stringhe, mentre solo *$attachments* è un *array*.

```
            $gm->send($to, $subj, $message,
                $cc, "", "", "", $attachments, false);
        } else {
            print("La connessione è fallita perché
                    ".$gm->lastActionStatus());
        }
}
```

Richiamiamo ora la *send()*: il codice mostra il passaggio dei parametri: è importante sottolineare come sia fondamentale l'ordine delle variabili e che, nel caso qualcuno non dovesse essere specificato, bisogna inserire una stringa vuota. L'unico parametro che non abbiamo definito con una variabile è il *false* alla fine. A cosa serve? A impostare l'invio della e-mail oppure a conservarla come *draft*: *false* sta per "*invia e-mail subito*" true invece come "*salvala in draft*".

CONCLUSIONI

La libreria è semplice quanto potente. e può essere utilizzata in diversi contesti. l'unica difficoltà potrebbe essere relativa alla comprensione delle costanti che fanno da interfaccia verso i comandi di Gmail, ma con un po' di pazienza la curva di apprendimento non sarà particolarmente complessa

Caterina Patrizia Morano

UN'APPLICAZIONE COMPLETA IN SOLI TRE FILE

Vediamo ora come inserire in un esempio completo la parte di codice che abbiamo appena descritto. L'applicazione permette di fare il login inserendo user name e password in un form per andare a prelevare e far vedere tutti i nomi dei contatti che abbiamo nella rubrica. Una volta visualizzati è possibile scegliere quelli a cui si intende inviare una e-mail ed effettuare la spedizione.
Vediamo come funziona nel dettaglio. L'applicazione è distribuita in tre file .php: test_ioprogrammo.php, contatti.php e invia_email.php. La prima visualizza due text-box per l'inserimento di user name e password, così da potersi loggare eventualmente anche con più account, qualora si voglia. Dopo aver cliccato su pulsante VAI! si viene richiama il file Contatti.php che si occupa del prelievo e della visualizzazione dei tutti i contatti che ci sono nella rubrica di gmail. I contatti vengono ordinati alfabeticamente e su ogni riga, oltre al radio button per la selezione, appare, in ordine, l'indirizzo e-mail e il nome del contatto: basta scegliere a chi spedire e cliccare sul tasto Invia le gmails. In questo modo verrà richiamato il terzo e ultimo file invia_email.php per la creazione dell'email e la spedizione.
Di seguito i tre files.

test_ioprogrammo.php

```
<html>
<head>
Proviamo la libreria Libgmailer
</head>
<body>
<table border="1">
<tr>
    <td>
        <form method="post"
            action="Contatti.php">

        <table>
         <tr>
            <td>Username</td><td><input
                type="text" name="user" /></td>

        </tr>
        <tr>
            <td>Password</td><td><input
                type="password" name="password"
                /></td>
        </tr>
        <tr>
            <td colspan="2"><input
                type="submit" value="Vai!" /></td>

        </tr>
        </table>
        </form>
    </td>
</tr>
</body>
</html>
```

contatti.php

```
<?php
include('include/libgmailer.php');
        if((isset($_POST['user'])) &&
            (isset($_POST['password']))) {

            $gm = new GMailer();
            if ($gm->created) {
                $gm-
                >setLoginInfo($_POST['user'],
                $_POST['password'], 0);

                if ($gm->connect()) {
                    $gm-
                    >fetchBox(GM_CONTACT, "all", "");
                    $snapshot = $gm-
                    >getSnapshot(GM_CONTACT);
                    $returned_emails =
                    $snapshot->contacts;
                } else {
                    die("Connessione
                    fallita: ".$gmailer->lastActionStatus());
                }
            } else {

                die("Non posso creare la
                classe: ".$gmailer->lastActionStatus());
            }
        }
?>
<html>
<body>
<form method="post"
            action="invia_email.php">
<table border="1">
<tr>
<td></td>
<td>Email</td>
<td>Nome Contatto</td>
</tr>
<?php
        foreach($returned_emails as $row){
            echo "<tr>";
            echo '<td><input
            type="checkbox" name="emails[]"
                value="'.$row['email'].'"
            checked="checked"/></td>';
            echo
            '<td>'.$row['email'].'</td>';
            echo
            '<td>'.$row['name'].'</td>';
            echo "</tr>\n\n";
        };
?>

</table>
<input type="submit" value="Invia le
                gmails" />
</form>
```

Invia_gmail.php

```
<?PHP
require("include/libgmailer.php");
 $gm = new GMailer();
        if (!$gm->created) {
            die("Non è stato possibile creare
            l'oggetto Gmailer: ".$gm-
            >lastActionStatus());
        } else {
            if ($gm->connect()) {
                $subj = "E-mail di
                prova";
                $message = "Ecco il
                testo dell'e-mail";
                $cc =
                "altrocontatto@indirizzo.it";
                $attachments =
                array("index.php");

                foreach
                ($_POST['emails'] as $email){
                    $gm-
>send($email, $subj, $message, $cc, "", "",
                "", $attachments, false);

                }
            }  else {
                print("La
                connessione è fallita perché ".$gm-
                >lastActionStatus());

            }
            print("Le gmail sono
            state mandate!!!");
        }
?>
```

PHP AD OGGETTI TECNICHE AVANZATE

I DESIGN PATTERN RAPPRESENTANO I PILASTRI FONDAMENTALI DI UN PROGETTO DESTINATO AL SUCCESSO. ESSI COSTITUISCONO IL CUORE PULSANTE DELLA PROGRAMMAZIONE ORIENTATA AGLI OGGETTI. VEDIAMO COME IMPLEMENTARNE ALCUNI IN PHP 5

In un articolo precedente abbiamo visto alcune delle caratteristiche avanzate di PHP 5. In particolare si è visto come implementare alcuni metodi speciali, come *__set*, *__get* e *__call*. Inoltre, abbiamo illustrato alcune caratteristiche orientate agli oggetti (OO) implementando due interfacce particolari, ossia *ArrayAccess* e *Iterator*. La prima per eseguire l'overload dell'operatore utilizzato dagli array, ossia la parentesi quadra [], mentre la seconda per rendere la nostra classe "traversabile".
In quest'articolo ci accingiamo ad esaminare il cuore della programmazione OO, ossia i design pattern. Infatti, visto che PHP ha introdotto la programmazione orientata agli oggetti (OOP) solo nella versione 5, molti programmatori PHP non sono a conoscenza di questi straordinari strumenti che rendono più semplice la vita del programmatore. Alla fine del presente articolo il lettore saprà:

- Cosa sono i design pattern.
- Come implementare, in PHP, alcuni dei pattern più comuni.

DESIGN PATTERN

Chiariamo subito che non è possibile analizzare, in modo approfondito, i design pattern in un singolo articolo. Basti pensare che vi sono interi libri dedicati all'argomento. Tuttavia, cercheremo di ridurre al minimo buona parte della teoria e rendere l'articolo quanto più pratico possibile. Chi vuole approfondire l'argomento, ad ogni modo, troverà diversi libri in commercio, come il classico della Gang of Four (**vedi box**).
Alla voce design pattern Wikipedia recita così: "Nell'ingegneria del software, un design pattern può essere definito 'una soluzione progettuale generale ad un problema ricorrente'. Esso non è una libreria o un componente di software riusabile, quanto una descrizione o un modello da applicare per risolvere un problema che può presentarsi in diverse situazioni durante la progettazione e lo sviluppo del software.
La differenza tra un algoritmo e un design pattern è che il primo risolve problemi computazionali, mentre il secondo è legato agli aspetti progettuali del software".
In pratica un design pattern è una soluzione rodata ad un problema ricorrente. Se siete degli esperti programmatori in un linguaggio orientato ad oggetti, è molto probabile che abbiate già usato un design pattern senza saperlo. Andiamo, ora, ad esaminare alcuni dei pattern più comuni e vediamo come implementarli in PHP.

STRATEGY PATTERN

Per presentare lo strategy pattern facciamo subito un esempio. Supponiamo di dover implementare diversi algoritmi di ordinamento e di usare l'uno o l'altro a seconda la situazione. In questo caso quale sarebbe la cosa più ovvia da fare? Nella programmazione procedurale verrebbe naturale implementare diverse funzioni (una per ogni algoritmo) con diversi nomi ed invocare quella adatta a seconda il contesto. Il fatto è che noi siamo programmatori che utilizzano l'OOP e quindi optiamo per una soluzione più flessibile. In pratica usiamo un'interfaccia (o una classe astratta) con un metodo, sort, e diverse classi che implementano tale interfaccia. Ci saranno tante implementazioni quanti algoritmi. Tutto ciò è illustrato dal class diagram di **figura 1**.

A livello di codice poi scriviamo qualcosa del genere:

```
$sorter = new BubbleSorter();
[...]
$sorter->sort($elements);
```

Supponiamo, ora, che un giorno ci accorgiamo di aver scelto un algoritmo di sorting meno efficiente. In questo caso ci basterà intervenire sulla

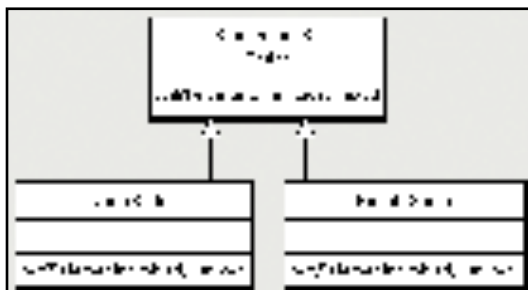



Fig. 1: Class diagram dello strategy pattern per il sorting.

creazione dell'istanza della classe che implementa l'algoritmo e la parte restante possiamo lasciarla inalterata. In termini di codice abbiamo:

```
$sorter = new QuickSorter();
// Tutto il resto rimane inalterato
```

Ciò che abbiamo fatto, nella letteratura dei design pattern, ha un nome ben preciso ed è lo strategy pattern. Secondo la Gang of Four, lo scopo dello strategy pattern è:
"Definire una famiglia di algoritmi, incapsulare ognuno di essi e renderli intercambiabili. Lo strategy permette di modificare i vari algoritmi in modo indipendente dal codice client che li usa".
In pratica, lo strategy si usa quando vi è la necessità di gestire diverse implementazioni di ciò che è, a livello concettuale, lo stesso algoritmo. Nel nostro esempio, concettualmente ci serviva ordinare un insieme di elementi. L'algoritmo, dunque, è lo stesso. Le implementazioni, tuttavia, sono più di una, ossia bubble sort, quick sort, merge sort e così via.
Terminiamo il paragrafo sullo strategy pattern con un interrogativo. Supponiamo che vi chiedano di implementare un algoritmo che permette di criptare un testo. Per fare ciò vi sono diversi algoritmi di crittografia (DES, Blowfish, ecc.). Ovviamente, deve esserci la possibilità di cambiare l'algoritmo di crittografia usato senza stravolgere il codice. Vi viene in mente qualche pattern di vostra conoscenza che vi permette di risolvere il problema in modo elegante?

FACTORY PATTERN

Il codice precedente, in realtà, viola un altro principio fondamentale di design. In pratica un progetto dovrebbe essere "aperto" alle estensioni ma "chiuso" alle modifiche. Sto parlando del cosiddetto Open-Closed Principle (OCP). Ad un primo acchito questo principio può sembrare contraddittorio. In realtà, se capito ed applicato può portare a progetti molto robusti ed ampiamente manutenibili. Come abbiamo violato l'OCP nell'esempio di sorting precedente? Lo abbiamo fatto in fase di creazione dell'istanza concernente l'algoritmo di ordinamento scelto. Infatti, in un caso reale la scelta dell'algoritmo sarebbe probabilmente gestita tramite una variabile globale o, preferibilmente, un file di configurazione o qualche altra tecnica che, ad ogni modo, comporterebbe un codice simile al seguente:

```
$sorter = null;
if($sortingAlgorithm == "quick")
{
  $sorter = new QuickSorter();
}
else if($sortingAlgorith == "bubble")
{
  $sorter = new BubbleSorter();
}
else
[...]

$sorter->sort($elements);
```

Possiamo riformulare l'OCP con la seguente frase: "Identifica la parte del tuo codice che cambia ed incapsulala". Nel codice precedente la parte che rimane uguale è l'invocazione del metodo *sort*. Tutta la parte precedente, invece, ha una buona probabilità di cambiare. Supponete, ad esempio, che in futuro si scopra un algoritmo di ordinamento più efficiente. In questo caso dovreste aggiungere un altro *else* al blocco appe-

NOTA

GANG OF FOUR (GOF)

La Gang of Four (banda dei quattro) o, in breve, GoF è il simpatico nomignolo con cui ci si riferisce spesso ai quattro autori del libro "Design Patterns: Elements of Reusable Object-Oriented Software" (ISBN 0-201-63361-2). Gli autori sono: Erich Gamma, Richard Helm, Ralph Johnson, e John Vlissides. Sono considerati i pionieri dei design pattern in quanto i primi a portare il concetto di pattern, già presente in architettura, nel mondo dell'ingegneria del software.

COME INIZIARE

Per provare il codice allegato a quest'articolo avete bisogno di PHP versione 5.2.* ed un Web server.
Il primo lo potete trovare qui:
http://www.php.net/

Come Web server io ho utilizzato Apache HTTP Server che potete trovare all'indirizzo:
http://httpd.apache.org/

Se utilizzate Windows e volete un'installazione molto semplice di PHP ed Apache Web server consiglio di utilizzare EasyPHP. Esso installa, in un colpo solo, PHP, Apache Web server, MySQL e PHPMyAdmin. Questi ultimi due non vi serviranno per gli esempi di questo articolo, ma vi saranno ad ogni modo utili per provare script che utilizzano il database MySQL. L'indirizzo per EasyPHP è il seguente:
http://easyphp.org/

Una volta installato ed avviato il server, potete provare gli script allegati utilizzando un URL simile al seguente:
http://localhost/ioprogrammo/php5/advanced-oop/design-patterns/Observer/client.php

Il precedente URL è valido se avete impostato Apache HTTP Server in ascolto sulla porta 80 (di default è così) e messo il codice allegato sotto il path ioProgrammo/php5/advanced-oop/design-patterns, che a sua volta va inserito sotto la root puntata da localhost. Se avete usato EasyPHP, il path completo sarà qualcosa del genere: C:\Programmi\EasyPHP 2.0b1\www\ioProgrammo\php5\advanced-oop\design-patterns\Observer

NOTA

RISORSE UTILI

PHP Home Page:
http://www.php.net/

Manuale online per PHP 5:
http://it.php.net/manual/en/language.oop5.php

Design Pattern (Wikipedia):
http://it.wikipedia.org/wiki/Design_pattern

na visto. Ciò che conviene fare, a questo punto, è isolare quel pezzo di codice (il blocco if-else) ed incapsularlo all'interno di una classe che, in termini di design pattern, prende il nome di classe factory. Così facendo, inoltre, rispettiamo anche il Single Responsibility Principle (SRP), ossia la classe factory ha la singola responsabilità di creare l'istanza del sorter opportuno. Assieme allo strategy, il factory pattern è quello che si utilizza più frequentemente.
Ecco il codice della nostra classe factory:

```
class SorterFactory
{
        private static $algorithms = array("bubble",
                                           "quick");

        static function create($name)
        {
                if (!in_array($name,
                        self::$algorithms))
                {
                        throw new
   Exception("L'algoritmo $name non è presente");
                }
                switch ($name)
                {
                        case "bubble": return
                        new BubbleSorter($name);
                        case "quick": return
                        new QuickSorter($name);
                }
        }
}
```

In pratica, il metodo *create*, prende in ingresso il nome dell'algoritmo da utilizzare. Se tale algoritmo non è presente lancia un'eccezione altrimenti, ritorna un'istanza dell'algoritmo desiderato.
Vediamo, ora, il codice client che utilizza la classe factory appena vista:

Fig. 2: Class diagram dell'observer pattern.

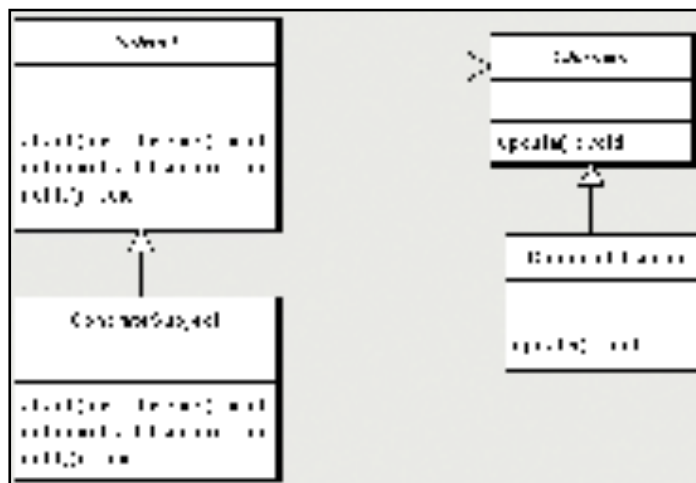
```
$elements = array("Alessandro Lacava", "Domenico
                        Morda'", "Antonino Catanese");
[...]
$sorter = SorterFactory::create("bubble");
$sorter->sort($elements);
[...]
```

Come potete vedere per usare, ad esempio, il quick sort vi basta cambiare il parametro *bubble* in *quick* e tutto il resto del codice può rimanere inalterato.
Inoltre, per rendere disponibile un altro algoritmo ci basterà implementare l'interfaccia *Sorter* ed aggiungere il codice di creazione in un solo punto, ossia nel metodo *create* della classe factory.

OBSERVER PATTERN

Un altro pattern molto utile è il cosiddetto Observer Pattern. Il class diagram semplificato di **figura 2** ne illustra il design.
Ed ecco la definizione:
"L'observer pattern definisce una dipendenza uno-a-molti tra oggetti, tale che quando un oggetto muta il suo stato, tutti gli oggetti che dipendono da esso sono notificati ed aggiornati automaticamente".
Formalismi a parte, tramite l'observer pattern possiamo creare un sistema event-driven in cui alcune entità dimostrano il loro interesse al verificarsi di un certo evento.
Un esempio classico è quello della Borsa. Vogliamo fare in modo che quando il prezzo di un'azione cambia alcuni investitori siano notificati. Il numero d'investitori da avvertire, ovviamente, è soggetto a cambiare nel tempo quindi dovremo sviluppare un sistema che, in modo flessibile, ci consentirà di aggiungere o rimuovere investitori.
L'observer pattern ci permette di risolvere il problema in modo elegante. Riferendoci alla **figura 2**, nel nostro esempio il subject astratto è la classe *Azione*, un subject concreto è rappresentato dalla classe *AzioneConcretaFittizia*, l'observer astratto è costituito dall'interfaccia *IInvestitore*, mentre l'observer concreto è rappresentato dalla classe *Investitore*. Vediamo di analizzare il codice che implementa tale pattern.

L'interfaccia *IInvestitore* è molto semplice:

```
interface IInvestitore
{
        function update(Azione $obj);
}
```

Essa espone un solo metodo, update. Notate che abbiamo usato un “hint” (**vedi box**) per specificare che il metodo accetta solo oggetti di tipo Azione. La classe concreta che implementa tale interfaccia è Investitore:

```
class Investitore implements IInvestitore
{
        private $nome;
        private $azione;

        function __construct($name)
        {
                $this->name = $name;
        }

  function update(Azione $azione)
  {
        echo "<b>Notifica per $this->name:</b>
                                                    ";
        echo "Il prezzo attuale di ". $azione-
       >getSimbolo(). " è " . $azione->getPrezzo();
        echo "<br />";
  }
}
```

Come potete notare, questa classe fornisce un’implementazione semplice del metodo update, ossia si limita a visualizzare un messaggio. Nella realtà questo metodo provvederebbe a comunicare il cambiamento all’investitore in modo tempestivo, ad esempio tramite un sms.

La classe astratta *Azione*, invece, è così implementata:

```
abstract class Azione
{
        protected $simbolo;
        protected $prezzo;
        protected $investitori = array();

        function __construct($simbolo, $prezzo)
        {
                $this->simbolo = $simbolo;
                $this->prezzo = $prezzo;
        }

        function attach(IInvestitore $investitore)
        {
                $this->investitori[] =
                                          $investitore;
        }

        function detatch(IInvestitore $investitore)
        {
                foreach($this->investitori as
                 $indice => $investitoreCorrente)
```

```
                {
                        if($investitoreCorrente
                             == $investitore)
                        {
array_splice($this->investitori, $indice, 1);
                              break;
                        }
                }
        }

        function notify()
        {
                foreach($this->investitori as
                        $investitoreDaAvvertire)
                {
$investitoreDaAvvertire->update($this);
                }
        }
    // Getter e Setter
    [...]
}
```

Gli attributi di questa classe sono *$simbolo*, *$prezzo* e *$investitori*. Il primo è una stringa che costituisce il simbolo dell’azione (ad esempio GOOG per Google). Il secondo, invece, è il prezzo al quale viene scambiata l’azione. L’attributo *$investitori* rappresenta un array di oggetti di tipi *IInvestitore*. Gli elementi di quest’array rappresentano gli osservatori che devono essere notificati quando il prezzo dell’azione cambia. A parte il costruttore, tale classe espone tre metodi:

- *attach*: aggiunge un investitore all’array
- *detatch*: rimuove un investitore dall’array
- *notify*: invoca il metodo *update* di ogni investitore, notificandolo così dell’avvenuto cambiamento del titolo.

Oltre a questi metodi vi sono i vari “getter” e “setter” che non mostriamo per brevità e semplicità d’implementazione. L’unico metodo che vogliamo analizzare è *setPrezzo*, poiché il cambiamento del prezzo è l’evento che determina la notifica dei vari investitori. Ecco il codice di *setPrezzo*:

HINT SUI PARAMETRI

PHP non è un linguaggio fortemente tipizzato, vale a dire non c’è bisogno, come in Java ad esempio, di dichiarare il tipo delle variabili. Tuttavia, possiamo fornire degli “hint” alle funzioni di modo che accettino parametri solo di un determinato tipo. E’ quello che abbiamo fatto con il metodo update dell’interfaccia IInvestitore, il quale accetta solo parametri di tipo Azione. In realtà PHP, internamente, esegue un controllo tramite l’operatore instanceof. Se tale controllo non va a buon fine, lo script fallisce.

NOTA

ORIGINI DEI PATTERN

Per quanto possa sembrare strano, i pattern non sono nati con l’ingegneria del software. Il padre universalmente riconosciuto dei pattern è Christopher Alexander, un architetto austriaco di nascita, ma cresciuto in Inghilterra. Anche se i suoi lavori riguardavano l’architettura, l’astrazione con cui introdusse il concetto di pattern fu tale che l’idea poté essere riutilizzata in vari campi, tra i quali l’ingegneria del software.

```
function setPrezzo($prezzo)
{
        $this->prezzo = $prezzo;
        $this->notify();
}
```

Come si può osservare, il metodo è abbastanza semplice. Si modifica il prezzo e si notificano tutti gli investitori tramite il metodo notify.
Ogni azione estenderà la classe astratta Azione. Ad esempio, la nostra AzioneConcretaFittizia è così implementata:

```
class AzioneConcretaFittizia extends Azione
{
        function __construct($simbolo, $prezzo)
        {
                parent::__construct($simbolo,
                                        $prezzo);
        }
}
```

Questa classe è così semplice che si limita ad invocare il costruttore della classe parent, ossia di Azione.
Un esempio di codice client che utilizza il nostro observer pattern è il seguente:

```
// Crea gli investitori
$investitore1 = new Investitore("Alessandro
                                        Lacava");
$investitore2 = new Investitore("Domentico Morda'");
$investitore3 = new Investitore("Antonino
                                        Catanese");
$investitore4 = new Investitore("Francesco
                                        Passaniti");

// Crea l'azione e registra gli investitori interessati
$azione = new AzioneConcretaFittizia("ACF", 70.69);
$azione->attach($investitore1);
$azione->attach($investitore2);
$azione->attach($investitore3);
$azione->attach($investitore4);

// Ad ogni cambio prezzo vengono notificati gli
                                        investitori
$azione->setPrezzo(69.96);

echo "<br />";

// Rimuove un investitore
$azione->detach($investitore4);

// Cambia il prezzo
$azione->setPrezzo(69.00);
```

In pratica, il codice appena visto crea quattro istanze di oggetti di tipo *Investitore*. In seguito crea un'istanza di *AzioneConcretaFittizia* e gli "attacca" i quattro investitori in precedenza creati. Non appena viene invocato il metodo *setPrezzo* sull'azione, il prezzo è modificato e gli observer (gli investitori) sono notificati del cambiamento. A scopo illustrativo viene poi rimosso un investitore e richiamato *setPrezzo* per riscatenare l'evento che causa la notifica degli investitori. L'output prodotto è il seguente:

```
Notifica per Alessandro Lacava: Il prezzo attuale di
                                        ACF è 69.96
Notifica per Domentico Morda': Il prezzo attuale di
                                        ACF è 69.96
Notifica per Antonino Catanese: Il prezzo attuale di
                                        ACF è 69.96
Notifica per Francesco Passaniti: Il prezzo attuale di
                                        ACF è 69.96
Notifica per Alessandro Lacava: Il prezzo attuale di
                                        ACF è 69
Notifica per Domentico Morda': Il prezzo attuale di
                                        ACF è 69
Notifica per Antonino Catanese: Il prezzo attuale di
                                        ACF è 69
```

CONCLUSIONI

In quest'articolo abbiamo introdotto il concetto di design pattern e ne abbiamo visti alcuni implementandoli in PHP. Un design pattern, tuttavia, è un concetto astratto. Un pattern può essere implementato in PHP5, Java, C# e qualsiasi altro linguaggio orientato agli oggetti. La loro potenza sta proprio in questo. Essi ci educano a progettare e sviluppare software robusto, flessibile e di qualità, a prescindere dal linguaggio OOP utilizzato. Ovviamente ciò che abbiamo visto deve essere inteso come una mera introduzione all'argomento. Lo scopo del presente articolo, infatti, era quello di accendere la lampadina nel lettore che, approfondendo i vari concetti esposti, potrà dare luce a software di qualità, flessibile e più facilmente manutenibile.

Alessandro Lacava

CODICE CLIENT

Nell'ambito dei design pattern, e non solo, si sente spesso parlare di "codice client". Con tale termine s'intende la parte di codice che utilizza il pattern. Bisogna fare attenzione, quindi, a non fraintendere pensando che il client sia un browser o qualcosa di simile. Ad esempio, anche quando si usano delle API, il codice che accede alle API viene spesso detto codice client.

NUVOLE ALL'ORIZZONTE

IL WEB 2.0 HA INTRODOTTO IL CONCETTO DI "TAG CLOUD". IN OGNI BLOG O SITO CHE SI RISPETTI ESISTE UNA RAPPRESENTAZIONE GRAFICA DEGLI ARGOMENTI CHE VEDE LE DISCUSSIONI PIÙ IN VOGA VISUALIZZATE CON UN FONT PIÙ GRANDE. VEDIAMO COME FARE IN .NET

Una delle caratteristiche apparse con l'avvento del Web 2.0, è la possibilità che può essere data ai visitatori o ai webmaster di un sito di classificare i contenuti in base a delle etichette o tag. Il fenomeno delle comunità Web, e dei siti che permettono una condivisione o aggregazione sociale, dimostra l'importanza che ha la condivisione delle informazioni. La popolarità di siti come del.icio.us e di Flickr che permettono il tagging dei contenuti d'altra parte mostra come il meccanismo delle nuvole di tag, o tag cloud è stato accolto dal popolo della rete in maniera molto favorevole. In un Tag Cloud è facile vedere, con una rapida occhiata quali argomenti sono più popolari o importanti di altri, basta notare quelli con dimensione maggiore. Qualsiasi lista di elementi rappresentabili sotto forma di testo o parole può usufruire di questo modo di rappresentazione.
L'articolo che vi accingete a leggere mostrerà come implementare un server control da utilizzare in una qualsiasi applicazione ASP.NET, sia congiuntamente ad una sorgente dati fisica come un database access, ma anche a partire da una collezione di oggetti business.

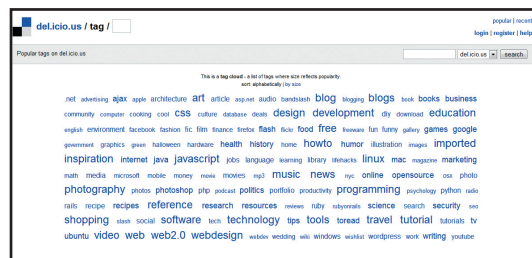



Fig. 1: tag cloud su del.icio.us

UTILIZZO DEL DATABINDING

Uno dei requisiti che vogliamo soddisfare nella progettazione del controllo TagCloud è la possibilità di poterlo legare ad una sorgente di dati, per esempio ad una tabella di database, oppure ad un controllo ObjectDataSource di oggetti .NET, e così via.
La classe CompositeDataBoundControl fa al caso nostro. Essa viene utilizzata come classe da cui derivare un controllo server che deve ricavare i dati da rappresentare da una sorgente. Un controllo composito inoltre è un controllo che viene costruito mediante composizione di altri controlli server, standard o personalizzato. Essa non viene mai istanziata direttamente, ma come faremo noi, la si usa come base da cui derivare un nuovo controllo DataBound; non mancano nel .NET Framwork degli esempi di classi derivate da essa molto importanti e praticamente utilizzati in ogni applicazione web ASP.NET, basti citare FormView e GridView.
Le classi che derivano da *CompositeDataBoundControl* devono fornire il proprio override del metodo CreateChildControls, che si occupa di creare la gerarchia visuale del controllo stesso, cioè l'insieme dei controlli figli che saranno contenuti nella collezione Controls.
Con un opportuno override di *CreateChildControls* verrà creata la nuvola di elementi, che dovranno possedere le proprietà che permettono di dimensionarli in base ad una importanza o peso, che permettono di navigare con un click su ogni elemento, che magari possano colorarsi in una gradazione che ne riassuma anch'essa l'importanza e quanto altro si vuole.

IL CONTROLLO TAGCLOUD

In questo paragrafo metteremo subito le mani in pasta, vedendo come progettare ed implementare un controllo server ASP.NET che ci permetterà di aggiungere un tag cloud ad un

qualunque sito web, scritto naturalmente nella stessa tecnologia.
Come detto la classe che rappresenta il controllo deve derivare dalla classe madre di ogni controllo data bound, e cioè da CompositeDataBoundControl. Un campo privato fondamentale della classe sarà naturalmente la collezione di elementi che rappresentano i tag da visualizzare nel controllo stesso, in questo caso utilizzeremo una Collection di oggetti TagCloudItem, classe mostrata subito dopo TagCloud:

```
public class TagCloud :
CompositeDataBoundControl
{

private Collection<TagCloudItem> m_items = new
                    Collection<TagCloudItem>();

[PersistenceMode(PersistenceMode.InnerProperty),
                    MergableProperty(false)]
 public Collection<TagCloudItem> Items
{
          get{return m_items;}
}
...
```

Passiamo poi a due campi statici che verranno utilizzati rispettivamente per determinare la dimensione del carattere del tag, e che quindi sarà maggiore per un "peso" maggiore assegnato al tag stesso, ed un campo per colorare il ag in maniera dipendente ancora dal peso, e che potete naturalmente personalizzare come meglio credete:

```
static string[] m_fontSizes = new string[]
{ "xx-small",
             "x-small",
             "small",
             "medium",
             "large",
             "x-large",
             "xx-large"
};

static string[] m_colors = new string[]
{  "#E3f555", "#bAb555", "#999555",
"#777555", "#555555", "#333555", "#111555"
};
```

Fig. 2: uml class diagram del controllo TagCloud

Continuando a parlare di pesi, passiamo al metodo *NormalizeWeight*, che permette di normalizzare un peso, e di ottenere l'indice da utilizzare per scegliere la dimensione di font dall'array m_fontSizes appena visto. La normalizzazione è necessaria per gestire dei pesi che possibilmente possono variare molto in magnitudine, e trasformarli in degli indici standard. Facciamo un esempio: potrebbero esserci dei tag che rappresentano categorie di articoli contenenti migliaia di articoli, mentre un'altra categoria potrebbe avere un solo articolo. Piuttosto che cercare di gestire ogni possibile grandezza, facciamo in modo che a determinare la magnitudine di un tag, sia una sorta di differenza "relativa", che normalizziamo dividendo ogni peso per la differenza fra il massimo valore possibile ed il minmo valore possibile. La differenza sarà naturalmente un valore sempre e comunque minore di 1 e maggiore di 0. Suddividendo l'intervallo [0,1] in dei sottointervalli, possiamo ricavarne differenti, in questo esempio 7, indici per accedere poi all'array m_fontSizes e m_colors.

```
private static int NormalizeWeight(double weight,
                    double min, double max)
{
        double normal = weight/(max-min);
        return (normal > 0.8) ? 7 :
                    (normal > 0.7) ? 6 :
                    (normal > 0.6) ? 5 :
                    (normal > 0.45) ? 4 :
                    (normal > 0.25) ? 3 :
                    (normal > 0.1) ? 2 :
                                   1;
}
```

Ogni tag può avere diverse proprietà, legabili ai dati sottostanti, ad esempio ad un database, o comunque utilizzabili per personalizzarne comportamento e aspetto. Per esempio le seguenti proprietà permettono di stabilire da quale altra proprietà di ogni oggetto della collezione Items debba essere ricavato il testo da utilizzare come collegamento ipertestuale ed

in che formato farlo:

```
publiic string DataHyperlinkField
{
        get
        {
            string val =
        ViewState["DataHyperlinkField"] as string;

            if (val != null)
                {
                        return val;
                }
                return String.Empty;
        }
        set
        {
        ViewState["DataHyperlinkField"] = value;
                if (this.Initialized)
                {
                    this.RequiresDataBinding = true;
                }
        }
}

[Category("Data")]
public string DataHyperlinkFormatString
{
        get
        {
            string val =
          ViewState["DataHyperlinkFormatString"]
                                        as string;

                if (val != null)
                {
                        return val;
                }
                return String.Empty;
        }
        set
        {
ViewState["DataHyperlinkFormatString"]
                                         = value;
                if (this.Initialized)
                {
this.RequiresDataBinding = true;
                }
        }
}
```

In maniera analoga vengono implementate le proprietà per determinare il peso, il titolo (cioè il tooltip), i dati dei tag.
Metodo fondamentale di ogni controllo Server di ASP.NET, e di cui bisogna creare un opportuno override, è il seguente CreateChildControls. In questo caso, per ogni tag contenuto nella collezione Items, creata a sua volta mediante il metodo CreateItemsFromData, viene creato un elemento HtmlAnchor, vale a dire un classico collegamento ipertestuale, sfruttando le proprietà precedenti per configurarli in maniera adeguata,

```
protected override int
        CreateChildControls(System.Collections.
                                    IEnumerable
                        dataSource, bool dataBinding)
{
        if (dataBinding && !this.DesignMode)
CreateItemsFromData(dataSource);
        double max = Max(Weights);
        double min = Min(Weights);
        int index = 0;

        foreach (TagCloudItem item in Items)
        {
                HtmlAnchor a = new
                                    HtmlAnchor();
                a.HRef =
String.IsNullOrEmpty(item.Href) ?
                            this.Page.ClientScript.
                  GetPostBackClientHyperlink(thi
                           s, index.ToString()) :
                            item.Href;
                a.InnerText = item.Text;
                a.Title = item.Title;

                int normalWeight =
        NormalizeWeight(item.Weight, min,max);
a.Style.Add(HtmlTextWriterStyle.FontSize,
                    m_fontSizes[normalWeight - 1]);
        a.Style.Add(HtmlTextWriterStyle.Color,
                        _colors[normalWeight-1]);
                this.Controls.Add(a);
                this.Controls.Add(new
                        LiteralControl(" "));
                index++;
        }
        if (this.DesignMode && Items.
                                    Count == 0)
        {
                HtmlAnchor a = new
                                    HtmlAnchor();
                a.InnerText = "TagCloud";
                this.Controls.Add(a);
        }
        return Items.Count;
}
```

Il già citato metodo *CreateItemsFromData*, crea, a partire da un oggetto che implementa l'interfaccia IEnumerable, la collezione di

LA CLASSE OBJECTDATASOURCE

La classe ObjectDataSource è un controllo data source di ASP.NET che rappresenta un oggetto con cui astrarre dai dati, o da utilizzare come interfaccia di accesso ai dati veri e propri. Un controllo ObjectDataSource può essere utilizzato in congiunzione ad un controllo grafico data-bound per visualizzare, modificare, cancellare dei dati attraverso una pagina Web, con poco codice o addirittura senza scriverne una sola riga.

elementi da rappresentare come tag.

```
private void
        CreateItemsFromData(System.Collections.
                IEnumerable dataSource)
{
        foreach (object data in dataSource)
        {
                TagCloudItem item = new
                        TagCloudItem();
                if
        (String.IsNullOrEmpty(this.DataHyperlinkField))
                {
                        if
        (String.IsNullOrEmpty(this.DataHyperlinkFormat
                        String))
                                item.Href =
                                String.Empty;
                        else
        String.Format(CultureInfo.CurrentCulture,
this.DataHyperlinkFormatString, new object[] {
                                data
                                });
                }
                else
                {
                item.Href = DataBinder.Eval(data,
this.DataHyperlinkField,
 this.DataHyperlinkFormatString);
                }
                if
        (!String.IsNullOrEmpty(this.DataTextField))
                {
                        item.Text =
                        DataBinder.Eval(data,
 this.DataTextField,
this.DataTextFormatString);
                }
                if
        (!String.IsNullOrEmpty(this.DataTitleField))
                {
                        item.Title =
                        DataBinder.Eval(data,
this.DataTitleField,
this.DataTitleFormatString);
                }
                if
        (!String.IsNullOrEmpty(this.DataWeightField))
                {
                        item.Weight =
                Convert.ToDouble(DataBinder.
                        GetPropertyValue(data,
 this.DataWeightField));
                }

                this.Items.Add(item);
        }
        }
    }
```

BIBLIOGRAFIA E SITOGRAFIA

MSDN Microsoft, http:/msdn.microsoft.com

Programming Microsoft ASP.NET, Dino Esposito, Microsoft Press

Developing Custom server Controls, msdn2.microsoft.com/en-us/library/zt27tfhy.aspx

GLI ELEMENTI DELLA NUVOLA

Ogni tag del controllo TagCloudControl è rappresentato da un'istanza della classe TagCloudItem, molto semplice nella sua struttura, e facilmente personalizzabile in base alle esigenze ed a queli dati si vogliano rapresentare ed utilizzare:

```
public class TagCloudItem
{
        private string m_text;
        private string m_title;
        private double m_weight;
        private string m_href;
public TagCloudItem()
        {
        }
        public TagCloudItem(string text, double
                                weight)
        {
                this.m_text = text;
                this.m_weight = weight;
        }
        public TagCloudItem(string text, double
                        weight, string href)
                : this(text, weight)
        {
                this.m_href = href;
        }
        public TagCloudItem(string text, double
                weight, string href, string title)
                : this(text, weight, href)
        {
                this.m_title = title;
        }
        public string Text
        {
                get
                {
                        return m_text;
                }
                set
                {
                        m_text = value;
                }
        }

        public string Href
        {
                get
                {
                        return m_href;
                }
```

```
            set
            {
                m_href = value;
            }
        }

        public string Title
        {
            get
            {
                return m_title;
            }
            set { m_title = value; }
        }

        public double Weight
        {
            get { return m_weight; }
            set { m_weight = value; }
        }
}
```

UTILIZZARE IL CONTROLLO

Il controllo così implementato è pronto ad essere utilizzato in una pagina ASP.NET. Basta avere una sorgente dati da usare per generare i tag del controllo. In questo primo esempio, sia per semplicità, ma anche per mostrare una diversa possibilità, sarà direttamente costruita la collection Items:

```
<dcweb:TagCloud ID=
            "tagCloud1" runat="server">
        <Items>
            <dcweb:TagCloudItem
                        Text="Item1"
                    Href="Test"
    Title="Default.aspx?tag=Item1" Weight="4"/>
            <dcweb:TagCloudItem
                        Text="Item2"
                    Href="Test"
Title="Default.aspx?tag=Item2" Weight="4" />
        </Items>
</dcweb:TagCloud>
```

Una possibilità più avanzata è quella di utilizzare un controllo ObjectDataSource. Costruendo per esempio una classe da utlizzare come sorgente dati ed una per rappresentare i tag, nella seguente maniera:

```
public class TagItem
{
        private int m_weight;
        private string m_name;
```

```
            public TagItem(string name, int
                                    weight)
            {
                this.m_name = name;
                this.m_weight =
                                    weight;
            }
            public string Name
            {
                get { return m_name;
                                        }
            }
            public int Weight
            {
                get { return
                                m_weight;
                                        }
            }

        public override bool Equals(object obj)
        {
            TagItem other = obj as TagItem;
            if (other == null)
                return false;
            return this.Name == other.Name;
        }
    }
    public class TagItemsSource
    {
            public TagItemsSource()
            {
            }
            public TagItem[] GetItems()
            {
                return new TagItem[]{
                new TagItem(".NET
                                    ",30) ,
                new TagItem("C#
                                    ",218) ,
        new TagItem("VB ",21) ,
                new TagItem("HTML
                                    ",4) ,
                new TagItem("Java
                                    ",69) ,
                new
            TagItem("JavaScript ",234) ,
                new TagItem("Perl ",6)
            new TagItem("PHP ",40)
                                        ,
                new TagItem("Ruby
                                    ",37) ,
            };
    }
}
```

Basterà a questo punto configurare l'ObjectDataSource come segue, all'interno del codice HTML della pagina:

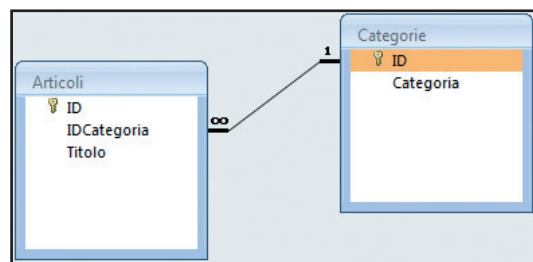



Fig. 3: le tabelle del database

```
<asp:ObjectDataSource ID="ItemsSource"
        runat="server" SelectMethod="GetItems"
                TypeName="ItemsSource" />
```

Al controllo TagCloud basterà indicare la sorgente di dati mediante la proprietà DataSourceID.
Le varie proprietà DataTextField, DataTitleField, e così via, verranno fatte corrispondere alle proprietà degli oggetti TagItem utilizzati come elementi da rappresentare nel controllo stesso:

```
<dcweb:TagCloud ID="tagCloud1" runat="server"
              DataSourceID="ItemsSource"
                DataTextField="Categoria"
                DataTitleField="Categoria"
        DataTitleFormatString="{0} articoli"
                   DataWeightField="Tag">
```

IL DATABASE

Per mostrare come ricavare i dati da un database realizzeremo una tabella Articoli ed una tabella Categorie, utilizzando Access per semplicità.
Le due tabelle sono correlate come mostrato in figura, vale a dire ogni articolo appartiene ad una ed una sola categoria. Contando poi quanti articoli fanno parte di una certa categoria, si utilizzerà il nome della categoria stessa come Tag, ed il numero di articoli di una data categoria come peso del tag stesso.
Aggiungendo alla pagina un controllo AccessDataSource, per ricavare tutti gli articoli presenti in archivio, basta una query come la seguente, mostrata all'interno della proprietà SelectCommand del controllo inserito all'interno della pagina aspx:

```
<asp:AccessDataSource ID="AccessDataSource1"
runat="server" DataFile="~/App_Data/items.mdb"
      SelectCommand="SELECT Articoli.ID,
          Articoli.IdCategoria, Articoli.Titolo,
   Categorie.Categoria FROM (Articoli INNER JOIN
         Categorie ON Articoli. IdCategoria =
                               Categorie.ID)">
</asp:AccessDataSource>
```

adesso, utilizzando il metodo Select di AccessDataSource, al caricamento della pagina, si può ricavare la collezione di TagItem da utilizzare impostando la proprietà DataSource del controllo TagCloud. Il metodo Select esegue la query presente nella proprietà SelectCommand e restituisce un oggetto che implementa l'interfaccia Ienumerable. Gli elementi di tale oggetto si possono scorrere mediante il metodo MoveNext. Ogni elemento è un oggetto DataRowView, che possiede tante colonne quante sono quelle indicate nella query:

```
IEnumerable results=
                AccessDataSource1.Select(new
                DataSourceSelectArguments());
List<TagItem> items = new List<TagItem>();
IEnumerator enumerator=
                results.GetEnumerator();
TagItem item;
DataRow row;
int index;
while(enumerator.MoveNext())
{
        row=(enumerator.Current as
                        DataRowView).Row;
        item=new
         TagItem(row["Categoria"].ToString(),1);

        if (items.Contains(item))
        {
            index=items.IndexOf(item);
            item=new
        TagItem(items[index].Name,items[index].
                                Weight+1);
            items.RemoveAt(index);
            items.Add(item);
        }
        else items.Add(item);
}
tagCloud1.Items.Clear();
tagCloud1.DataTextField = "Name";
tagCloud1.DataTitleField = "Weight";
tagCloud1.DataTitleFormatString = "{0} articoli
                        sull'argomento";
tagCloud1.DataWeightField = "Weight";
tagCloud1.DataHyperlinkField = "Name";
tagCloud1.DataHyperlinkFormatString =
                  "Default.aspx?tag={0}";
tagCloud1.DataSource = items;
```

per ogni elemento, vale a dire per ogni articolo, viene letta la categoria, ed incrementato il numero, cioè il peso, del tag. Se un item con

L'AUTORE

Antonio Pelleriti, ingegnere informatico, sviluppa software da più di dieci anni e si occupa di .NET sin dalla prima versione Beta. È chief software architect di DynamiCode s.r.l., Software Factory in cui progetta e sviluppa soluzioni custom ed in outsourcing (www.dynamicode.it). Può essere contattato per suggerimenti, critiche o chiarimenti all'indirizzo e-mail antonio.pelleriti@dotnetarchitects.it

una certa categoria non esiste ancora, esso viene rimosso dalla Lista e reinserito dopo averne incrementato di uno la quantità.
Alla fine del ciclo che legge tutta l'enumerazione di articoli trovati, vengono impostate le proprietà del controllo TagCloud, mostrando come testo di ogni Tag la proprietà Name del TagItem, che in questo caso sarà il nome della categoria, mentre come titolo viene ancora una volta mostrato il testo "{0} articoli sull'argomento", sostituendo al parametro {0}: il numero di articoli presenti nella categoria.
La figura seguente mostra il risultato

CONCLUSIONI

Nell'articolo si è visto come implementare un server control di ASP.NET per realizzare l'ormai onnipresente funzionalità dei Tag Cloud, le nuvole di Tag che permettono di riassumere e visualizzare visivamente, gli argomenti più popolari di un sito web o di un blog. Dopo aver realizzato il controllo abbiamo visto come utilizzarlo all'interno di un normale progetto web, utilizzando come sorgente dati sia un Object Data Source, quindi una collezione di oggetti .NET, sia un database access dal quale leggere i dati con cui costruire gli oggetti da rappresentare nel controllo. Si potrebbero effettuare ancora alcune migliorie ad esempio progettando un backend per la gestione manuale dei pesi e l'immissione/rimozione di argomenti. Ma questo lo lasciamo alla vostra fantasia...

Antonio Pelleriti

***Fig. 4:** il nostro tagcloud in funzione*

CREARE ADDON PER INTERNET EXPLORER

MOLTI PENSANO CHE LO SVILUPPO DI ADD-ON PER IE SIA RISERVATO A PROGRAMMATORI C++ , VEDIAMO COME È INVECE POSSIBILE AGGIUNGERE NUOVE FUNZIONALITÀ AL BROWSER SEMPLICEMENTE ATTRAVERSO DEI SEMPLICI SCRIPT

Lo sviluppo di funzionalità aggiuntive per i browser Web trova indubbiamente in Firefox la piattaforma più "amichevole" per il programmatore: con un po' di XML e di javascript è possibile ottenere dei risultati veramente notevoli (basti pensare a "Firebug" che è ormai diventato un must per il debugging per gli sviluppatori web). In Internet Explorer le cose non sono, a prima vista, così facili. Analizzando la documentazione in merito ci si imbatte infatti in delle API di basso livello, quasi esclusivamente riservate a programmatori C++ (anche se, come vedremo in un prossimo articolo, è possibile utilizzare anche il .NET Framework). In definitiva quindi sembrerebbe che lo sviluppo di funzionalità aggiuntive per IE sia un lavoro che richiede un impegno non certamente "amatoriale". Tuttavia questo assunto non è del tutto vero. In realtà quello che non tutti sanno è che anche soltanto utilizzando javascript è possibile sviluppare degli add-on per IE.
In particolare vedremo come sviluppare, attraverso gli script, nuovi comandi per il menu principale ed il menu contestuale di IE.

UN PO' DI TEORIA

Per programmare un add-on attraverso uno script c'è ben poca teoria da conoscere, tuttavia è comunque necessaria qualche nozione di base.
Per prima cosa bisogna sapere che la configurazione di un nuovo add-on è tutta contenuta nel registro di configurazione di Windows.
Occorre andare alla chiave del registro di configurazione

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\
                    Internet Explorer\Extensions
```

All'interno della chiave creare una sotto-chiave che ha per nome una stringa GUID che identifica in modo univoco il nostro Add-On.
Sotto quest'ultima chiave è necessario creare:

- Un nuovo valore *String* di nome *CLSID* con valore *{1FBA04EE-3024-11d2-8F1F-0000F87ABD16}* (che indica che si tratta di una voce di menu).
- Un nuovo valore *String* di nome *MenuText* che sarà il testo della voce.
- Un nuovo valore *String* di nome *Script* che conterrà il percorso assoluto dello script

Una cosa importante da ricordare è che il file contenente lo script deve avere estensione .htm (non .js come potrebbe apparire più logico) si tratta praticamente di un file HTML senza interfaccia, che contiene solo il tag <*script*>, ad esempio:

```
<script language="javascript"
                    type="text/javascript">
/* ... codice */
</script>
```

CONFIGURAZIONE DI UNA NUOVA VOCE

L'aggiunta di una nuova voce al menu contestuale è più o meno simile. Si tratta qui di agire sulla

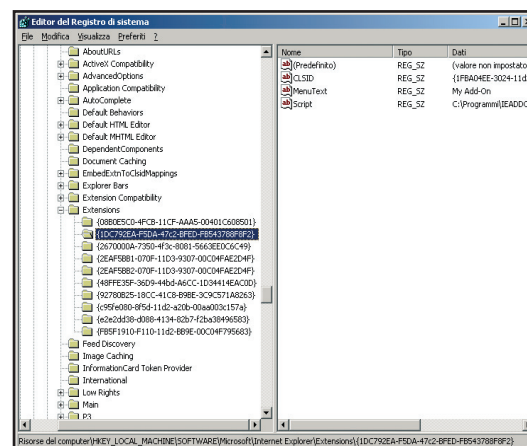



Figura 1: configurazione di un nuovo Add-On nel registry di Windows

chiave di registro:

```
HKEY_CURRENT_USER\Software\Microsoft\Internet
                                        Explorer\MenuExt
```

Qui creeremo una nuova sotto-chiave il cui nome rappresenterà il testo della voce di menu, ad esempio:

```
HKEY_CURRENT_USER\Software\Microsoft\Internet
                              Explorer\MenuExt\Mio Menu
```

Nel valore di default della chiave inseriremo il percorso assoluto dello script
Opzionalmente potremo inserire nella nuova chiave un valore DWORD chiamato *Contexts* che determina in quali situazioni far comparire la voce nel menu contestuale il valore potrà essere uno di questi (combinabili anche con OR logico):

Contesto	Valore
Default	**0x1**
Immagini	**0x2**
Controlli	**0x4**
Tabelle	**0x8**
Selezione di testo	**0x10**
Link	**0x20**

Se poi vogliamo che il file HTML contenente lo script si apra in una nuova finestra di dialogo modale (come se fosse stato richiamato con *showModalDialog*) occorre creare il valore DWORD dal nome *Flags* impostato ad 1.

REALIZZARE UN PRIMO ADD-ON PER IL MENU CONTESTUALE

Come abbiamo visto la configurazione di nuove voci di menu è abbastanza semplice, vediamo quindi di creare il nostro primo Add-On utilizzando gli script.
Il nostro Add-On sarà qualcosa di più del classico "Hello World" : si attiverà sul menu contestuale e, in una finestra di dialogo, visualizzerà il codice HTML corrispondente alla selezione sulla pagina.
Il codice sarà tutto contenuto in un semplice file HTML (ShowSource.htm) che contiene un elemento <PRE> ed uno script.
Lo script avrà il compito di :

- trovare la selezione corrente della pagina,
- estrarne il codice HTML,
- trasformare i tag < e > nelle entità HTML *<* e *>* (altrimenti non si vedrebbe il codice, ma l'output)
- Inserire il tutto nell'elemento *<PRE>*.

Nello scrivere il codice ci troviamo di fronte ad una prima difficoltà : se il codice viene eseguito in una finestra di dialogo, quindi diversa dalla pagina del browser, mentre lo script si riferisce ad oggetti presenti nell'altra pagina, come fare ad ottenere il riferimento all'oggetto *window* della pagina principale?
Per fortuna il problema è facilmente risolvibile, la finestra principale è infatti referenziata dalla proprietà *menuArguments* dell'oggetto *external* della finestra corrente.
Il codice quindi, alla fine, si presenta così:

```
<html>
<head>
        <title>Selection Source</title>
        <link rel="STYLESHEET" type="text/css"
                                href="star-light.css">
</head>
<body>
        <pre id="htmlContainer" class="html">
        </pre>
        <script language="javascript"
                            type="text/javascript">
/*
riferimento a finestra principale
*/
                var win =
                          external.menuArguments;
                var doc = win.document;
/*
Selezione corrente
*/
            var sel = doc.selection;
                var rng = sel.createRange();
                var src = rng.htmlText;
/*
Se la selezione corrente non esiste prende tutto il
                                    codice del BODY
*/
                if (src.length <= 0) src =
                                   doc.body.outerHTML;
                src =
    src.replace(/</g,'&lt;').replace(/>/g,'&gt;<br/>')
                htmlContainer.innerHTML=src

        </script>
</body>
</html>
```

In questo caso, per la formattazione del codice HTML della finestra abbiamo utilizzato un foglio di stile esterno; il nostro script può infatti fare riferimento a CSS e SCRIPT esterni che ri-

COS'È UN GUID

Il GUID (Globally Unique Identifier, identificatore unico globale) è un numero utilizzato per identificare in maniera univoca un componente software. Anche se il GUID non è sicuramente univoco il numero possibile di combinazioni (2128) è talmente elevato da evitare una coincidenza di due codici in un sistema.
Nel sistema Windows il GUID viene utilizzato soprattutto nel Registro di Configurazione (Registry) di sistema. Maggiori informazioni possono essere reperite su http://it.wikipedia.org/wiki/GUID

siedano nella stessa posizione (o in cartelle interne) del file stesso.
A differenza delle voci del menu principale, che non hanno finestre di dialogo, le voci di menu contestuali possono attivare finestre di dialogo, quindi nel file HTML che andremo a predisporre il markup è relativo a quest'ultime.

CONFIGURAZIONE

Una volta creato il nostro file HTML occorre configurarlo nel Registry di Windows.
Per prima cosa dovremo creare una chiave nella posizione :

```
HKEY_CURRENT_USER\Software\Microsoft\Internet
                    Explorer\MenuExt
```

Attenzione perché il nome della chiave sarà quello che apparirà come voce del menu contestuale. Nel nostro caso il nome sarà Show Source, quindi creeremo la chiave :

```
HKEY_CURRENT_USER\Software\Microsoft\Internet
            Explorer\MenuExt\Show Source
```

Come valore di default (Predefinito) della chiave assegniamo la path completa del file di codice che abbiamo editato (nel nostro caso sarà *C:\Programmi\IEADDONS\Test\ShowSource.htm*). Adesso, poiché vogliamo che il nostro Add-on apra una finestra di dialogo, creiamo un valore di tipo DWORD chiamato *Flags* e gli assegniamo il valore 1.

TEST E DEBUG

Avviamo IE, o chiudiamolo e riavviamolo se fosse stato già aperto.
Andiamo a selezionare una porzione di pagina (testo, immagini ecc...) e, attivando il menu contestuale con il tasto destro, vedremo comparire la nuova voce *Show Source* (**figura 2**).
Cliccando sulla voce quindi verrà eseguito il nostro Add-on in una nuova finestra di dialogo come quella visibile in **figura 3**. Per il debug invece la cosa interessante da sapere è che ogni volta che cambiamo qualcosa nel file di codice non è necessario riavviare Internet Explorer.

REALIZZARE UN PRIMO ADD-ON PER IL MENU PRINCIPALE

Gli Add-On del menu contestuale si prestano

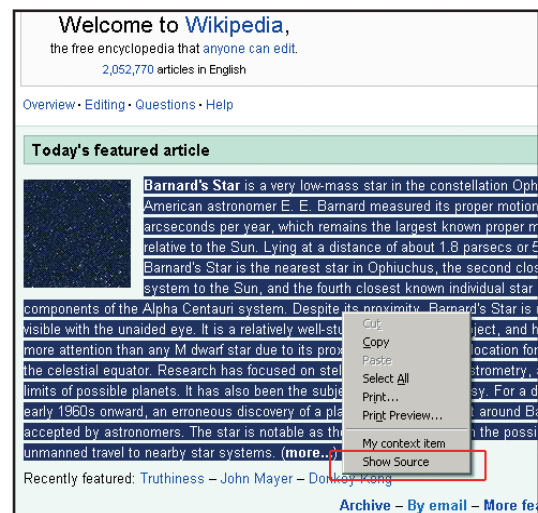



Figura 2: voce nel menu contestuale di IE

particolarmente per azioni da applicare alla singola selezione, quelli al menu principale invece sono più indicati per azioni relative all'intera pagina.
Passiamo quindi a creare un Add-On per il menu principale che ha la semplice funzione di rivelare la struttura del documento corrente contornando le tabelle di blu, le celle di rosa e i DIV di verde (cosa che può essere utile allo sviluppatore per comprendere il layout della pagina).

IL CODICE

Anche qui creiamo un nuovo file HTML, che chiameremo *ShowStructure.htm*, che però contiene solo il tag <*script*> (ogni altro markup sarebbe infatti inutile visto che l'Add-On non viene visualizzato in una finestra ma si applica direttamente alla pagina). Il codice anche qui è semplice:

- si ottiene il riferimento alla pagina attraverso *external.menuArguments* e, da questa, al documento
- si crea una semplice funzione che imposta il bordo di un elemento con un dato colore
- si ottengono gli elementi corrispondenti ai Tag che vogliamo contornare
- si applica la funzione agli insiemi di elementi in precedenza ricavati

Tutto il codice contenuto in *ShowStructure.htm* sarà quindi:

```
<script language="javascript" type="text/javascript">
/*
Oggetto window della pagina corrente
*/
var parentwin = external.menuArguments;
/*
```

```
Oggetto document della pagina corrente
*/
var doc = parentwin.document;
/*
Funzione per applicare bordi colorati
*/
function applyBorder(elements,color){
        for(var i=0;i<elements.length;i++){
                var node = elements[i];
                node.style.border = "1px solid " +
                                                color;
        }
}
/*
Insiemi di elementi
*/
var tables = doc.getElementsByTagName("TABLE");
var cells = doc.getElementsByTagName("TD");
var divs = doc.getElementsByTagName("DIV");
/*
Applicazione della funzione
*/
applyBorder(tables,"#3300ff");
applyBorder(cells,"#ff0099");
applyBorder(divs,"#33ff00");
</script>
```

CONFIGURAZIONE

La configurazione dell'Add-On è leggermente diversa rispetto a quella vista per il menu contestuale. Per prima cosa dovremo scegliere una GUID (Globally Unique Identifier) che identifichi in modo univoco il nostro Add-on, per farlo se abbiamo Visual Studio possiamo usare il generatore che troviamo tra gli strumenti, *guidgen.exe*, altrimenti possiamo usare uno dei tanti tool online (uno di essi è anche nel mio sito all'indirizzo www.smelzo.it/guidgen).
Quindi dovremo creare una chiave nella posizione :

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\
                    Internet Explorer\Extensions
```

Utilizzando come nome la GUID prescelta, ad esempio:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\
Internet Explorer\Extensions\{1DC792EA-F5DA-47c2-
                              BFED-FB543788F8F2}
```

All'interno della chiave creiamo:

- Un valore STRING chiamato CLSID, questo valore deve corrispondere a {1FBA04EE-3024-11d2-8F1F-0000F87ABD16} (che indica che si tratta di un'estensione al menu principale).
- Poi creiamo un altro valore STRING che chiamiamo MenuText , questo valore contiene il testo della voce di menu, nel nostro caso sarà "Show Structure".
- Infine creiamo un ulteriore valore STRING chiamato Script che contiene la path completa del file di codice che abbiamo editato (nel nostro caso sarà C:\Programmi\IEADDONS\Test\ShowStructure.htm).

TEST

Terminata la configurazione chiudiamo IE se era in esecuzione e riavviamolo e apriamo una pagina qualsiasi.
A questo punto se andiamo nel menu principale alla voce Strumenti/Tools noteremo che è comparsa una nuova voce, Show Structure appunto. Attivando questa voce notiamo che lo script viene eseguito ed il layout viene evidenziato.

CONCLUSIONI

In questo articolo abbiamo visto il modo più semplice di personalizzare IE, ovvero mediante l'utilizzo di semplici script Javascript.
Anche se l'utilizzo delle API consente cose ovviamente di fare cose più complesse, attraverso questa tecnica comunque si possono costruire degli Add-On non banali con un impegno di tempo ragionevole.

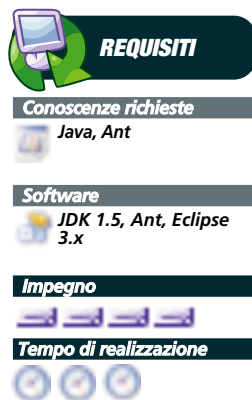
Francesco Smelzo

Figura 3: nuova voce nel menu principale di IE

DOCUMENTAZIONE

La documentazione sulle estensioni di IE mediante Script non è particolarmente abbondante, qualcosa può essere comunque reperito all'indirizzo http://msdn2.microsoft.com/en-us/library/aa753616.aspx.

JMATTER SEMPLIFICA LO SVILUPPO IN JAVA

LE APPLICAZIONI SOFTWARE SPESSO COMPRENDONO UN SET DI FUNZIONALITÀ MOLTO SIMILI; PERCHÈ IMPLEMENTARLE OGNI VOLTA DA ZERO? OGGI TUTTO CIÒ SI PUO EVITARE E JMATTER TI DA UNA MANO A FARE TUTTO PIÙ VELOCEMENTE

L'ingegneria del software negli ultimi anni ha assistito al consolidamento di tecniche e metodologie di sviluppo che hanno facilitato sotto tutti gli aspetti i progetti di piccole, medie e grandi dimensioni. Il mondo degli sviluppatori non si è fatto mancare proprio nulla; gli approcci così detti "Agili" hanno rimpiazzato le tecniche classiche di sviluppo, altrettanto numerose sono state le novità che hanno scosso il nostro mondo.

Nonostante tutti questi miglioramenti è impossibile negare che molti dettagli implementativi riguardano aspetti ripetitivi della programmazione stessa. Immaginiamo il caso di un'applicazione desktop, a prescindere dal dominio applicativo, il progettista prima e il programmatore dopo dovranno sempre prendere delle decisioni che non dipendono (o meglio non dovrebbero dipendere) dal dal problema in sè stesso. Come strutturare l'interfaccia grafica? Usare un ORM per il mapping dei dati oppure limitarsi all'utilizzo di JDBC? ORM sì, ma quale? Una volta prese queste decisioni, si presentano altri processi decisionali ancora non dipendenti dal contesto. In quale directory mettere le immagini che compaiono sull'interfaccia grafica? Oltre a queste decisioni, anche alcune funzionalità di base dovranno per forza essere implementate; si pensi all'autenticazione degli utenti e ai ruoli da assegnare loro.

Tali decisioni, tali funzionalità ed i conseguenti dettagli implementativi incidono sui tempi di realizzazione in modo drammaticamente pesante. A volte si arriva a soglie che toccano il 90%. In questo articolo studieremo e prenderemo in esame un framework che ci facilita nella realizzazione di applicazioni desktop, il suo utilizzo è così proficuo che consente di concentrare i nostri sforzi quasi esclusivamente sul dominio di business dell'applicazione stessa senza occuparci dello strato "fisico" che risiede sotto la logica implementativa. In pratica JMatter fornirà la struttura software sulla quale applicare la logica di business del progetto. A prescindere dal contesto avremo a disposizione un'interfaccia grafica ben strutturata e definita. Sapremo già in quale directory posizionare le immagini dei pulsanti, anche il nome da dare a tali immagini sarà già stabilito dal framework. Questi sono solo piccoli esempi delle facility studiate e messe a disposizione da JMatter. Tra poco potremo goderne i benefici attraverso una vera e propria full-immersion grazie alla quale andremo a sviluppare un'applicazione discretamente complessa.

INSTALLARE JMATTER

L'utilizzo di JMatter prevede dei requisiti software dei quali deve essere fornito il nostro *enviroment* di sviluppo; in particolare è prevista la presenza di un JDK 1.5 e di Ant. Acquisiti questi requisiti, bisogna scaricare dall'url www.jmatter.org/pages/download il framework in formato .zip. Supponiamo di scompattare il tutto sotto "C:\", avremo una directory C:\jamtter-<dataRealese> sotto la quale possiamo trovare tutto ciò che ci serve. Infatti possiamo notare la presenza di tre directory: *jmatter*, *modules* e *demo-apps*. La prima contiene le librerie pricipali su cui si basa il framework, la seconda le librerie di supporto mentre nella terza ci sono esempi completi e funzionanti.

PRONTI, VIA!

Come spesso accade in questa rivista il modo migliore per imparare l'utilizzo di un framework di sviluppo è quello di vederlo all'opera in un progetto concreto. L'obiettivo di questo progetto è quello di realizzare la "gestione delle letture", una sorta di schedario delle nostre fonti di informazione. Ovviamente la nostra applicazione prevede una fase di login che devono effettuare sia gli utenti semplici sia il cosiddetto amministratore. Per meglio percepire le peculiarità degli strumenti messi a disposizione da questo framework procederemo in modo incrementale, ovvero aggiungeremo funzionalità poco per volta evidenziandone i dettagli implementativi. Per prima cosa dobbiamo costruire l'infrastruttura iniziale del progetto; per fare ciò spostiamoci sotto *C:\jmatter-<dataRealese>\jmatter* e digitiamo il seguente comando:

ant new-project-ui

Figura 1: La form per la creazione di nuovi progetti.

Così facendo apparirà una un piccolo form come riprodotto in fig 1.
Bisogna inserire il nome del progetto (il nostro lo chiameremo GiElle, Gestione Letture), la directory del workspace sotto la quale creare il progetto ed infine il tipo di applicazione *Standalone* o *Dependent*; nel nostro caso sceglieremo la prima delle due opzioni. Compilati i tre campi e cliccato il pulsante "*Create Project*", si avvierà un processo abbastanza veloce che creerà una directory "GiElle" nel nostro workspace di lavoro contenete il necessario per la realizzazione del progetto stesso. In particolare sotto *lib* troviamo altre directory contenenti tutti i jar da cui JMatter dipende. Se avessimo scelto *Dependent* non avremmo avuto tali jar ed avremmo dovuto aggiustare a nostro piacimento il classpath dell'applicazione stessa. La directory *resources* contiene alcune risorse tra cui le più importanti sono le immagini contenute sotto *images* (torneremo a parlare di questa directory).
La directory *src* è quella in cui dovremo inserire le classi dell'applicazione, c'è da notare che tale directory contiene già alcuni file xml la cui semantica sarà discussa in seguito. Sotto GiElle troviamo anche un file build.xml che contiene *target* Ant pronti per l'uso.

INDIVIDUARE IL DOMINIO

Concentriamoci adesso sul dominio di business (model) della nostra applicazione. Ovvero cerchiamo di individuare le entità che tratteremo e di cui il nostro progetto si servirà durante i suoi processi funzionali. L'entità centrale è rappresentata da Lettura che contiene i seguenti campi:

- nome: nome della lettura
- location: dove si trova
- descrizione: descrizione della lettura
- parole_chiavi: parole chiavi da utilizzare nelle ricerche
- argomenti: argomenti trattati dalla lettura
- tipologia: libro, rivista, pagina web

I primi tre campi li possiamo considerare semplici e ricondurli al tipo *String* (per il momento), anche *parole_chiavi* lo possiamo considerare un'istanza di *String* ma stabiliamo da adesso che lo valorizzeremo con stringhe separate da spazi bianchi. Un discorso diverso va fatto per gli ultimi due campi, infatti *argomenti* e *tipologia* possono essere ricondotti a tipi composti. Il campo *Tipologia* per il momento lo consideriamo composto dal solo attributo *nome*. *Argomenti* invece ci induce a pensare ad un insieme di oggetti di tipo *Argomento*. Quest'ultimo avrà la seguente composizione:

- nome: nome dell'argomento
- descrizione: descrizione dell'argomento
- letture: le letture che trattano questo argomento

Poiché un argomento può essere trattato in diverse letture ed analogamente una lettura può trattare diversi argomenti, possiamo stabilire fin da ora che la relazione tra *Lettura* ed *Argomento* è *molti a molti*. Mentre poiché una lettura afferisce ad una sola tipologia (o è un libro o è un articolo di una rivista e cosi via) possiamo stabilire che la relazione che lega *Tipologia* e *Lettura* è *uno a molti*.
Questo lavoro di analisi è fondamentale per l'utilizzo di JMatter, infatti da qui a poco impareremo che basta tradurre questi requisiti in linguaggio Java per implementare la maggior parte delle funzionalità del software in questione. Iniziamo a scrivere il codice necessario per la realizzazione della nostra applicazione; per prima cosa vediamo come è strutturata la classe Tipologia:

```
@Persist
public class Tipologia extends
                              AbstractComplexEObject{
   private final StringEO nome = new StringEO();
   public static final String[] fieldOrder = {"nome"};
   public Tipologia() {}
   public Title title() {
     return nome.title();
   }
   public StringEO getNome() {
     return nome;
   }
}
```

Già da questi primi passi ci si accorge che JMatter usa delle convenzioni per definire il meta-model (una sorta di descrizione del modello) con cui costruire l'applicazione. L'annotation *@Persist* indica al fra-

NOTA

DOPPIA LICENZA

JMatter sembra aver pensato proprio a tutto, infatti il framework è distribuito con doppia licenza: GPL per gli amanti dell'open source, e commerciale ad un prezzo di 500$ per gli sviluppatori che non vogliono rendere pubblico il codice implementato.

NOTA

SCRIPT DI LANCIO

Il file Ant build.xml generato sulla root del nostro progetto è corredato da diversi target; il target shellscript è motlo utile, infatti una volta eseguito crea un file run.bat che consente di lanciare l'applicativo fuori da ambienti di sviluppo come Eclipse.

mework che le istanze della classe in questione saranno memorizzate su DB ed esporranno funzionalità di lettura e scrittura. In pratica questa annotation consente al framework di capire che l'entità in questione va mappata tramite Hibernate su una tabella del DB.
Ogni entità del dominio deve essere rappresentata da una classe che estende *AbstractComplexEObject*; questa classe contiene l'insieme dei metodi che avremmo dovuto ridefinire per ogni entità; basti pensare al mapping su DB e a come viene visualizzata attraverso l'interfaccia grafica. In precedenza abbiamo detto che le istanze di Tipologia avranno un unico campo *nome*; ecco JMatter ha riscritto un insieme di tipi base (Atomic types) che replicano i tipi base del linguaggio Java. Il campo *nome* è dichiarato *final* ed è inizializzato immediatamente; ad esso corrisponde un metodo *getNome* che ritorna un oggetto di tipo *StringEO*. Tutto ciò consente di avere un campo mai null e modificabile dall'esterno solo invocando il metodo *setValue(...)* sull'oggetto restituito dal suo *getter*

Figura 2: I tool messi a disposizione come target Ant.

(getNome). Ogni entità prevede anche un campo con la seguente *signature*:

```
public static final String[] fieldOrder= {"nome"};
```

L'array di stringhe in questione serve a definire l'ordine con cui verranno visualizzati i campi di una classe sull'interfaccia grafica; anche qui è prevista una corrispondenza tra i valori stringa assegnati agli item dell'array e i nomi dei campi della classe. Non ci resta che descrivere la semantica del metodo *title*; questo metodo serve ad identificare con una stringa le istanze di questa classe. Questo metodo è dichiarato abstract nella superclasse quindi siamo obbligati a definirlo per tutte le entità del nostro modello.

CODICE? GRAZIE, MA NON TROPPO

Aumentiamo leggermente la complessità ed occupiamoci di capire come è strutturata la classe Argomento.

```
@Persist
public class Argomento extends
                                  AbstractComplexEObject{
   private final StringEO nome = new StringEO();
   private final StringEO descrizione = new
                                           StringEO();
   public static final String[] fieldOrder =
                   {"nome","descrizione ","letture"};
   private final RelationalList letture = new
                          RelationalList(Lettura.class);
   public static final Class lettureType = Lettura.class;
   public static final int lettureRelationType =
              PersistenceMechanism.MANY_TO_MANY;
   public static final String lettureInverseFieldName =
                                          "argomenti";
   public RelationalList getLetture(){
     return letture;
   }

   public Argomento(){}
   public Title title(){
     return nome.title();
   }
   public StringEO getNome() {
            return nome;
   }
   public StringEO getDescrizione() {
            return descrizione;
   }
}
```

Anche in questo caso possiamo notare la presenza dell'annotation *@Persist* ed il fatto che la nostra

classe estende *AbstractComplexEObject*; per entrambi però la semantica è simile a quella analizzata per la classe *Tipologia*. Come abbiamo visto quando abbiamo definito il modello dell'applicazione, questa classe è composta dai campi *nome*, *descrizione* e *letture*. Per i primi due c'è poco da discutere in quanto, come visto nel caso precedente, si usa il metodo *get<nomeCampo>* per accedere al valore, mentre si usa il metodo *setValue* per modificarne il valore. In definitiva la vera novità è rappresentata dal campo *letture*. Anch'esso è dichiarato *private final* e prevede la definizione di un *getter*, il tipo con cui è dichiarato è *RelationalList*. Questa classe ci fa già intuire che si tratta di una lista di oggetti specializzata per mantenere relazioni tra oggetti appartenenti ad un determinato modello. Il tipo di oggetti contenuti in questa lista è specificato sia nel costruttore sia nella dichiarazione

```
public static final Class lettureType = Lettura.class;
```

Come possiamo notare il nome attribuito ai vari campi della classe non è casuale, infatti anche in questo caso, la parte iniziale (letture) è identica al nome della lista di cui si vuole specificare il tipo. Questo *modus operandi* può risultare a volte poco elegante ed un po' ridondante; l'autore del framework è consapevole di talune forzature sintattiche, precisa che sono dovute a vari fattori quali la datazione del framework anteriore alla definizione dei *generics*, e alla velocità con cui le API si sono evolute lasciando poco spazio al refactoring. Andando avanti nell'analisi del codice, possiamo osservare che la costante intera *lettureRelationType* serve a definire che la relazione definita dal campo *letture* è di tipo molti a molti; come nel caso precedente siamo vincolati nel definire il nome della costante. Altro codice degno di nota è il seguente:

```
public static final String lettureInverseFieldName =
                                          "argomenti";
```

Sfruttando la reflection, quando il framework incontra questa dichiarazione capisce che la relazione è mantenuta anche in senso inverso, ovvero nella classe Lettura ci sarà un campo *argomenti* che manterrà una relazione con la classe corrente.

LA CLASSE LETTURA

E' giunto il momento di dare uno sguardo alla classe principale che la fa da protagonista nella nostra applicazione; ovviamente stiamo parlando della classe Lettura.

```
@Persist
public class Lettura extends
                              AbstractComplexEObject{
  private final StringEO nome = new StringEO();
  private final StringEO location = new StringEO();
  private Tipologia tipologia= new Tipologia();
  private final TextEO descrizione = new TextEO();
  private final StringEO parole_chiavi = new
                                          StringEO();
  private final RelationalList argomenti = new
                         RelationalList(Argomento.class);
```

In questa prima parte non troviamo niente di particolare da annotare; infatti non abbiamo fatto altro che rispettare le convenzioni viste già in precedenza. Abbiamo definito una classe le cui istanze saranno rese persistenti dal framework, ed abbiamo definito i campi che ne determinano il contenuto di informazioni. Un piccolo appunto va fatto per il campo *descrizione*, come possiamo notare è stato dichiarato di tipo TextEO. Questa classe si differenzia da StringEO per due motivi principali; il primo è che l'interfaccia grafica riserva un *textfield* per l'input dei dati di tipo StringEO, mentre agli oggetti istanze di TextEO un *textarea*. L'altra differenza si riscontra nella memorizzazione su DB, per il tipo StringEO viene utilizzato un *var-*

JMATTER E DB

JMatter utilizza Hibernate per mappare le proprie classi sul DB, quindi è ovvio che ha la possibilità di utilizzare tutti i DB supportati da Hibernate. Di *default* però (per esempio nalla nostra applicazione) utilizza H2 database. Questo DB viene utilizzato in modalità *embedded*; ovvero non c'è bisogno di lanciare nessun server per il DB. Il DB infatti viene lanciato contestualmente all'esecuzione dell'applicazione. Per cambiare a proprio piacimento la tipologia di DB in cui memorizzare i dati, è sufficiente editare il file di proprietà *hibernate.properties*. Tale file è reperibile sotto la directory <GiElle>/resources.

Figura 3: GiElle e la sua veste grafica.

char(255) mentre per TextEO un tipo *text* (o affine, dipende dal DB). Come avevamo stabilito in fase di *design* ad ogni istanza di Lettura è associata una ed una sola istanza della classe Tipologia (relazione 1:1); mentre come avevamo stabilito per la classe Argomento, in questa classe abbiamo definito una lista di argomenti che serve a mantenere la relazione molti a molti.

```
public static final Class argomentiType =
        Argomento.class;
public static final int argomentiRelationType =
        PersistenceMechanism.MANY_TO_MANY;

public static String argomentiInverseFieldName =
        "letture";

public static final String[] fieldOrder = {"nome",
"location","tipologia","descrizione","parole_chiavi","ar
gomenti"};
```

Anche il precedente codice Java non presenta novità rispetto alle conoscenze acquisite precedentemente; in pratica le prime tre istruzioni servono a specificare il contenuto dell'oggetto argomenti. Mentre l'array di string lo ribadiamo serve a stabilire l'ordine con cui vengono presentati i dati nelle form di modifica dei dati. Non ci stancheremo mai di dire che i valori delle costanti passati nell'array corrispondono e devono corrispondere ai nomi delle variabili istanze di classe.

```
public Lettura(){}
public Title title(){
  return nome.title();
}
public RelationalList getArgomenti() {
  return this.argomenti;
}

public Tipologia getTipologia() {
        return tipologia;
}

public void setTipologia(Tipologia tipologia) {
        Tipologia tipO=this.tipologia;
        this.tipologia = tipologia;
        this.firePropertyChange("tipologia", tipO,
                tipologia);
}

public TextEO getDescrizione() {
        return descrizione;
}

public StringEO getLocation() {
        return location;
}

public StringEO getNome() {
        return nome;
}

public StringEO getParole_chiavi() {
        return parole_chiavi;
}
}
```

Eccoci giunti all'ultima parte del codice della classe Lettura, i metodi in questione sono per la maggior parte dei casi i *getters* dei campi istanze di classe. Un caso particolare è rappresentato dal metodo *setTipologia*. Poiché tipologia è legata alla nostra classe con una relazione 1:1 è necessario notificare al framework il cambiamento delle istanze di tale classe. Se analizziamo minuziosamente il suo codice

```
Tipologia tipO=this.tipologia;
this.tipologia = tipologia;
```

possiamo notare che prima viene recuperato e memorizzato nell'oggetto *tipO* il vecchio valore, dopo di che viene aggiornato il valore corrente con l'oggetto passato come parametro. Infine

```
this.firePropertyChange("tipologia", tipO, tipologia);
```

viene notificato all'*engine* di eventi sottostante che la proprietà tipologia è cambiata, allo stesso *engine* vengono notificati sia il vecchio valore sia quello nuovo.

PURA MAGIA!

Una volta scritte le classi che rappresentano il modello del nostro dominio applicativo il lavoro del programmatore è quasi finito. Questa affermazione, per quanto folle possa sembrare, è il vero vanto di cui si fregia JMatter. Ci manca davvero poco per vedere la nostra applicazione funzionante.
Sotto la directory *src* esiste il file *class-list.xml*. Per rendere l'interfaccia grafica più fruibile bisogna modificare questo file. Nel file XML troviamo due occorrenze del tag *folder*, la prima contenente un tag *name* valorizzato a Model ed il secondo con il tag *name* valorizzato ad Admin. Bisogna editare il primo di questi tag in modo che sia valorizzato come di seguito.

```
...
<folder>
  <name>Model</name>
  <items>
    <type>roby.jmatter.gielle.Tipologia</type>
    <type>roby.jmatter.gielle.Lettura</type>
```

AGGIUNGERE NUOVI UTENTI

Nel nostro esempio abbiamo dato per scontato che l'utilizzatore si autentichi sempre come admin; è comunque possibile aggiungere nuovi utenti ed assegnare loro nuovi ruoli. L'interfaccia è molto intuitiva, basta loggarsi come admin aprire la sezione Admin e cliccare sull'icona Utenti e successivamente su Ruoli.

```
<type>roby.jmatter.gielle.Argomento</type>
  </items>
</folder>
...
```

In pratica questo file serve a specificare il posizionamento delle entità del modello nel menu di sinistra (*class-bar*). Nel nostro caso abbiamo cancellato le entità che sono già definite nel framework e le abbiamo rimpiazzate con quelle implementate in questo esempio. Finalmente siamo pronti per vedere la nostra opera in esecuzione. Dobbiamo prima lanciare il comando

```
ant schema-export
```

Questo comando trasforma le entità da noi implementate in tabelle e successivamente in tutti i file di configurazione Hibernate necessari per utilizzarle. Successivamente con

```
ant run
```

possiamo lanciare la nostra applicazione. Per prima cosa vediamo che compare un form di login in cui ci vengono richieste le credenziali per accedere al sistema. Utilizzando la coppia *admin admin* per username e password possiamo loggarci al sistema. In figura 3 possiamo notare l'aspetto dell'applicazione; abbiamo una *class-bar* sulla sinistra, un normale menu in alto a sinistra ed una zona inizialmente vuota al centro. Il *class-bar* è l'elemento con cui familiarizzare subito, infatti in esso sono definite due sezioni Model e Admin. La prima sezione l'abbiamo configurata in modo da presentare solo tre icone corrispondenti alle entità da noi introdotte, invece la sezione Admin è quella riservata alla configurazione dell'applicativo stesso. Ogni icona della sezione Model consente di eseguire un set di funzioni predefinite, tra le quali:

1. Elenca: visualizza tutte le entità presenti sul DB
2. Nuovo: consente di creare una nuova entità
3. Cerca: cerca le entità presenti sul DB tramite un form di immissione dati

Esistono poi altre funzionalità che per una prima introduzione al framework possono essere trascurate. Se ritorniamo un attimo alle icone visualizzate nella sezione Model possiamo notare che, contrariamente a quelle della sezione Admin, hanno tutte la stessa icona (quella di default), inoltre sono etichettate con nomi errati (Tipologias, Letturas, Argomentos) ; questo errore è dovuto al fatto che il framework utilizza la sintassi inglese per determinare la versione al plurale dei nomi. Per ovviare a ciò ritorniamo ai

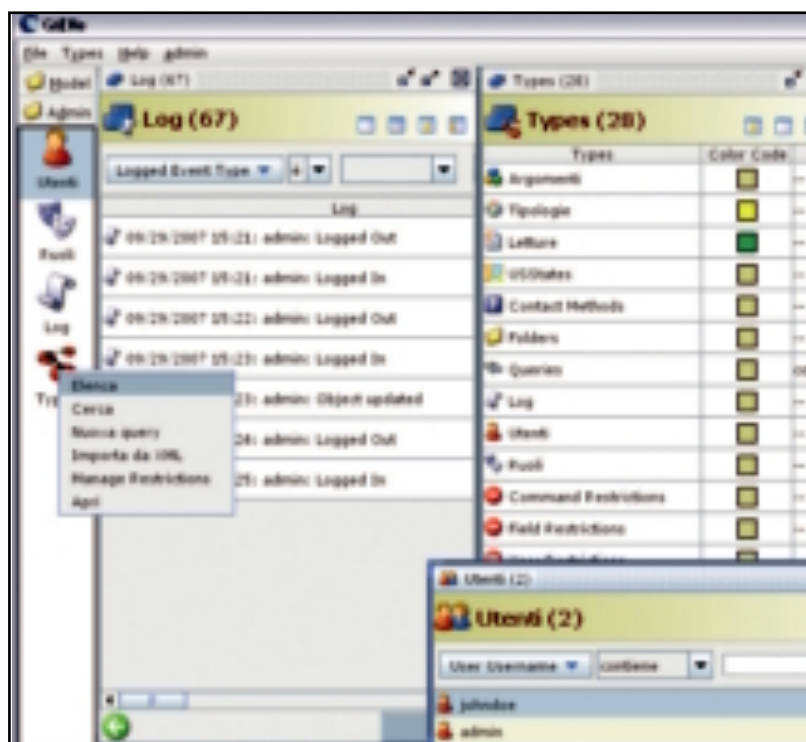



Figura 4: Ecco la parte di amministrazione della nostra applicazione.

VALIDAZIONE

Se si fa qualche prova con l'applicazione che abbiamo creato, ci si accorge che abbiamo trascurato la validazione dei dati; supponiamo di voler introdurre una nuova Tipologia. Allora ci posizioniamo sulla relativa icona, tasto destro e scegliamo il comando Nuovo. A questo punto ci appare una form di immissione dati dove dovremmo inserire il nome della tipologia. Cosa succede se clicchiamo su Salva senza aver inserito nessun nome? Succede che il sistema accetta lo stesso questo inserimento e visualizza una Tipologia vuota. Allora la feature che dobbiamo aggiungere deve obbligare l'utente ad inserire almeno un carattere nella casella nome. Ci sono due modalità per inserire tale comportamento. Per utilizzare la prima basta andare sotto la directory <GiElle>/resources ed editare il file model-metadata.properties. Essendo un file di properties è sufficiente aggiungere il seguente codice:

```
Tipologia.nome.required=true
```

In questo modo, all'atto del salvataggio, il sistema ci segnalerà in modo standard che il campo nome non può essere vuoto. Per implementare una strategia di validazione customizzata bisogna scrivere un po' di codice; è sufficiente effettuare l'override del metodo validate nelle entità implementate. Nel nostro caso dobbiamo aggiungere il seguente codice alla classe Tipologia.

```
public int validate(){
  String
        s=this.getNome().stringValue();
  if(s==null || s.trim().equals("")){
fireValidationException("Inserisci un
                  nome non vuoto!");
    return 1;
  }
  return 0;
}
```

Il metodo effettua un controllo sul campo nome dell'oggetto, se il suo valore è null o uguale ad una serie di spazi bianchi segnala all'utente un messaggio ad-hoc. Lo stato della validazione è determinato dal valore ritornato dal metodo; 0 segnala che la validazione è andata a buon fine mentre un qualsiasi altro intero segnala che la validazione non ha dato esito positivo.

sorgenti delle tre classi ed aggiungiamo il seguente codice:

```
public static String pluralName() { return "Letture"; }
```

Ovviamente per la classe Tipologia usiamo la stringa "Tipologie" e per la classe Argomento la stringa "Argomenti". Per quanto riguarda invece l'utilizzo di icone distinte e personalizzate per le varie entità la soluzione è ancora più semplice. Infatti basta munirsi di due immagini per entità aventi il nome così formato:

```
<nome_entità>16.png
<nome_entità>32.png
```

La prima avrà dimensione 16 x 16 e la seconda 32 x 32. Questi file vanno inseriti all'interno della directory <GiElle>/resources/images, sarà compito del framework caricarle ed associarle alle entità giuste.
Un altro miglioramento che possiamo facilmente apportare alla veste grafica è l'utilizzo di un splash screen in fase di caricamento dell'applicazione. Anche in questo caso è sufficiente munirci di un file splash.png (anche i formati gif e jpg vanno bene) e posizionarlo nella directory <GiElle>/resources/images.

L'AUTORE

Roberto Sidoti è Ingegnere Informatico, in passato si è occupato di servizi VoIP; attualmente è Functional Designer per Herzum Software, una multinazionale di consulenza specializzata nello sviluppo di componenti per applicazioni SOA. Negli ultimi mesi lavora per il progetto FourthCodex, che è una piattaforma di ultima generazione per la gestione di contenuti semantici.

COMANDI AGGIUNTIVI

JMAtter potrebbe somigliare ad un bel framework che prevede un set di funzionalità preconfezionate, che però poco si presta all'implementazione di funzionalità che vanno oltre le semplici operazioni su DB. Questa convinzione va subito smentita, JMatter consente allo sviluppatore di definire delle funzionalità customizzate che si possono aggiungere alle varie entità del dominio applicativo. Vista la natura didattica dell'articolo ci limiteremo a definire un semplice comando con poca valenza semantica, infatti lo utilizzeremo per cancellare il campo descrizione della classe Lettura. Non è oggetto di questo articolo ma va detto che il framework offre la possibilità di accedere tramite HQL ai dati sul DB.

```
@Cmd(mnemonic='c')
public String Cancella_Descrizione(CommandInfo
                                        cmdInfo){
  this.descrizione.setValue("");
  this.save();
  return "La descrizione è stata cancellata";
}
```

La sintassi prevista è semplicissima, per prima cosa va definito un metodo il cui nome corrisponde all'etichetta che intendiamo visualizzare sulla form. In questo caso il carattere "_" sarà rimpiazzato dal carattere " ". Per questo metodo va definita una annotation *@cmd* che indica al framework che il metodo successivo implementa la gestione di un *custom command*. Il carattere "c" sarà utilizzato come *shortcut* per l'esecuzione del comando. Di *default* il framework associa i custom command alla form di lettura (visualizzazione) degli oggetti; per utilizzare comandi personalizzati in altre form è possibile aggiungere porzioni di codice simili al seguente.

```
static{
  ComplexType type =
                    ComplexType.forClass(Lettura.class);
  Command cmd = new
         Lettura().command("Cancella_Descrizione ");
  type.commands(ReadState.class).remove(cmd);
  type.commands(EditState.class).add(cmd);
}
```

La semantica è molto semplice, una volta individuato il comando introdotto in precedenza lo si rimuove dallo stato ReadState e lo si aggiunge per lo stato EditState.

CONCLUSIONI

In questo articolo abbiamo imparato ad utilizzare un framework che segue i dettami del pattern Nacked Object. Abbiamo visto che il framework è utilissimo sia alla prototipazione sia alla realizzazione vera e propria di progetti software. Certamente il suo utilizzo per certi aspetti imbriglia la fantasia degli sviluppatori; questo fattore però va valutato positivamente in quanto gli stessi potranno concentrarsi solo sulla parte di business logic senza perdere tempo sulle altre parti. Un altro aspetto che vi consigliamo di tenere in considerazione è la continuità con cui questo framework viene aggiornato; infatti allo stato attuale il framework è efficiente e funzionante ma vi sono un paio di *macro-feature* che la comunità, che lo segue, si è riproposta di implementare. Si tratta della possibilità di avere un client web Ajax (magari utilizzando Echo2) e di estendere l'attuale architettura client/server magari utilizzando il remoting di Spring. E' inutile sottolineare che la conoscenza di tale framework e il fascino della programmazione in Java possano essere un buon input per indurre i nostri lettori alla partecipazione attiva in questo progetto *open source*.

Roberto Sidoti

REALIZZARE UN INSTANT MESSENGER

LO SVILUPPO DI UN INSTANT MESSENGER PUÒ ESSERE UN'ATTIVITÀ MOLTO INTERESSANTE CHE CI PERMETTERÀ DI ADDENTRARCI NEI MEANDRI DEL NETWORKING E DI REALIZZARE UN'APPLICAZIONE DI IMMEDIATO UTILIZZO PRATICO

Tutte le comunicazioni che avvengono su internet sono basate sul protocollo TCP/IP che è lo standard “de facto” nel campo del networking. Quando si parla di TCP/IP ciò che è fondamentale tenere presente è il concetto di “strati”, o all'inglese “layers”, che sta alla base del funzionamento del TCP/IP. In effetti lo stesso nome TCP/IP deriva dagli ultimi due layers del protocollo stesso. Mettiamo che A voglia inviare una mail a B: l'applicazione (liv. 7) di A propagherà il messaggio usando il layer sottostante (liv. 6), e che a sua volta userà il layer inferiore, fino ad arrivare alla comunicazione sul mezzo fisico. A sua volta dalla parte di B il messaggio verrà propagato dal liv. 1 fino al liv. 7. In altre parole ciò significa che ogni qual volta abbiamo la necessità di sviluppare un'applicazione che debba comunicare attraverso internet dovremo semplicemente “agganciarci” al layer più in alto dello stack TCP/IP, ossia il TCP (gli altri protocolli “fratelli” del TCP, come UDP o ICMP, esulano dagli scopi di quest'articolo ma il concetto sarebbe comunque il medesimo).

Una volta appurato ciò il passo successivo dovrebbe essere quello di chiedersi:

Come si fa nel concreto ad attaccarsi a questo stack di protocollo di comunicazione?

Ho voluto appositamente utilizzare il verbo attaccarsi perché esso rende bene l'idea su cosa si fondano i socket. Il termine socket in inglese significa “presa” , “attacco”.

E' in effetti come una porta di comunicazione, concettualmente simile ad una presa elettrica: qualsiasi cosa progettata per comunicare tramite il protocollo standard TCP/IP, può collegarsi ad un socket e comunicare tramite questa porta di comunicazione, così come un qualsiasi apparecchio elettrico alimentato con la corrente 220 può collegarsi ad una presa elettrica a muro e sfruttare la tensione che la rete di distribuzione elettrica mette a disposizione. Nel mondo dei socket potremmo dire che la rete di distribuzione è internet stessa ed invece dell'elettricità, nella rete viaggiano pacchetti TCP/IP.

Grazie a questo meccanismo quindi possiamo completamente disinteressarci a cosa si celi dietro ad un socket (proprio come non ci poniamo il problema di cosa ci sia dietro la presa dell'elettricità, almeno fin quando tutto funziona!), rendendo quindi del tutto trasparente l'esistenza stessa dello stack.

Scendendo più nel dettaglio di come sono fatti questi socket potremmo iniziare con l'affermare che essi sono identificabili da una coppia di parametri:

- l'indirizzo ip del host
- il numero di porta TCP

solitamente la sintassi prevede che questi elementi che compongono una coppia univoca siano separati dai due punti (:). Ad esempio se volessimo aprire un socket verso il nostro web server dovremmo specificare l'indirizzo 127.0.0.1:80.

Come avrete capito anche da quest'ultimo esempio, è necessario che i due interlocutori comunichino sulla stessa porta contemporaneamente! Altrimenti chi manda il messaggio troverebbe la “porta chiusa” dell'altro e viceversa. Da qui nasce l'esigenza del paradigma client – server.

JAVA ED I SOCKET

Per concludere la sintesi necessaria a fornire gli elementi per lo sviluppo della nostra applicazione vediamo in che modo Java supporti i socket. Le API di Java prevedono proprio questi due interlocutori (client – server). Rivediamo quindi brevemente le due classi in questione e come esse comunicano. Riporto l'esempio del server che risponde rimandando indietro i messaggi inviatigli dal client.

```
public class EchoServer implements Runnable {
        public static final int PORT = 4422;
        private ServerSocket sSocket = null;
        ..........
public void run() {
                try {
                        sSocket = new
                            ServerSocket(PORT);
                }
                .....
                Socket clientSocket = null;
```

```
PrintWriter out = null;
BufferedReader in = null;
running = true;
try {
    while (loop) {
        clientSocket =
sSocket.accept();
        out = new
PrintWriter(clientSocket.getOutputStream(),true);
        in = new
BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
        String line;
        while ((line =
in.readLine()) != null)
out.println("Server response: "+ line);

    }
} catch (IOException e) {
    e.printStackTrace();
}
....
System.err.println("Server is
down");
}
```

In poche parole si istanza un oggetto di tipo *ServerSocket* indicando al costruttore su quale porta debba rimanere in ascolto. Il momento in cui però esso si mette effettivamente in ascolto è solo al momento dell'invocazione del metodo *accept()* che restituisce il socket del client che ha effettuato la richiesta. La comunicazione con quest'ultimo viene quindi realizzata semplicemente leggendo e scrivendo sugli stream di input ed output del client stesso. Infatti vista da lato client la comunicazione avviene nel seguente modo:

```
public static void main(String[] args) throws
IOException {
Socket echoSocket = null;
    PrintWriter out = null;
    BufferedReader in = null;
    try {
        echoSocket = new
Socket("127.0.0.1", PORT);
        out = new
PrintWriter(echoSocket.getOutputStream(), true);
        in = new BufferedReader(new
InputStreamReader(
echoSocket.getInputStream()));
    }
    .....
BufferedReader stdIn = new BufferedReader(
new
InputStreamReader(System.in));
String userInput;
while ((userInput =
```

```
stdIn.readLine()) != null) {
    if(userInput.equals(""))
    {
serverSocket.stop();
        break;
    }
    out.println(userInput);
System.out.println(in.readLine());
}
```

Cioè un client non fa altro che puntare ad una ben precisa porta di un ben preciso indirizzo e scrivere e leggere sui propri stream, sapendo che dall'altra parte c'è un server in ascolto.

IL NOSTRO INSTANT MESSENGER

Dovremmo ormai quindi avere tutte le informazioni e conoscenze per affrontare lo sviluppo dell'applicazione che vogliamo realizzare: un Instant Messenger!
Sicuramente è naturale che al mondo non è che ci sia bisogno di un ulteriore programma di chat, ma affrontando questo progetto abbiamo l'opportunità di toccare con mano una considerevole serie di tecniche di networking riutilizzabili poi nei più disparati contesti.
D'altra parte poiché un intera infrastruttura per un programma completo di messaggistica richiederebbe probabilmente ben più di una serie di articoli lasceremo fuori quegli aspetti che non riguardano strettamente il networking, in particolare non ci occuperemo di:

1. Modulo di autenticazione con password
2. Gestione del database degli utenti e delle liste contatti

Ora che abbiamo definito bene i confine della nostra applicazione possiamo iniziare a sviluppare le parti che la comporranno.

UN'ARCHITETTURA P2P

L'esempio introduttivo ai socket mostrava l'utilizzo di un'architettura client / server; nel caso di un sistema di messaggistica diretto le cose dovranno essere diverse, infatti sarebbe più appropriato adottare un'architettura peer to peer (p2p) Cominciamo con un'analisi generale per poi scendere nel dettaglio di ogni singolo modulo.

- Quando si invia un messaggio tramite il bottone “send”) la GUI si comporterà da client ed aprirà un socket verso l'indirizzo dell'altro utente

- In fase di ricezione sarà necessario avviare il server come demone e metterlo in ascolto su una ben determinata porta. Una volta accettata una connessione da parte del client socket dovrà spedire il messaggio ricevuto alla GUI del proprio stack.
- La comunicazione per questa applicazione è sicuramente di complessità maggiore rispetto a quella del nostro primo esempio in cui il server si limitava a "fare l'eco" al client. Da qui nasce quindi l'esigenza di prevedere un protocollo che si occupi dell'elaborazione del messaggio. Tale compito sarà assolto da una classe che lavorerà in stretta coppia con il server.
- Il messaggio così elaborato sarà posto su una coda messa a disposizione dalla GUI. Uno spooler si occuperà di controllare tale coda ed andare a scrivere il messaggio sulla GUI stessa.

Come potete vedere un messaggio fa un bel giro prima di arrivare dall'altro utente. Seguiamo allora tale percorso proprio attraverso il codice.

SPEDIRE UN MESSAGGIO DALL'ALTRA PARTE

Appena aperta l'applicazione viene chiesto di inserire un nickname che identificherà l'utente durante tutta la sessione di comunicazione (figura 4). Tale nome viene passato al costruttore della GUI principale che lo memorizza come campo privato.

```
public class IMGUI extends javax.swing.JFrame {
.....
public IMGUI(String name) {
        super();
        userName = name;
        initGUI();
        queue = new
            SynchronousQueue<Message>();
.......
```

Viene inoltre istanziata una coda sincrona dove verrano recapitati i messaggi in entrata per l'utente. La classe *SynchronousQueue* è un generic e quindi il relativo oggetto è stato proprio creato specificando la classe Message come argomento del template.
Oltre a questa coda sarà necessario creare ed avviare i thread per il server socket e lo spooling della coda:

```
        try {
            serverDaemon = new
        Thread(new IMServer(4422,queue));
serverDaemon.setDaemon(true);
            serverDaemon.start();
        } catch (IOException e) {
JOptionPane.showMessageDialog(this,
            e.getMessage(), "ERROR",
        JOptionPane.ERROR_MESSAGE);
        }
        spooler = new Thread(new
                        Runnable(){
            public void run() {
                spool();
            }
        });
        spooler.setDaemon(true);
        spooler.start();
        ........
```

Il codice sottostante il bottone send è molto simile alla parte client vista nell'esempio precedente:

```
private void jButtonSendMouseClicked(MouseEvent
                                    evt) {
        if(client == null)
            client = new
    IMClient(userName,jTextFieldHost.getText(),Integer.p
        arseInt(jTextFieldPort.getText()));
        .....
        String message =
            jTextFieldMessage.getText();
        try {
            client.send(message);
jEditorPaneChat.getDocument().insertString(jEditor
    PaneChat.getDocument().getLength(), "[Me] " +
                message+"\n", null);
jTextFieldMessage.setText("");
        } catch (UnknownHostException
                                e) {
JOptionPane.showMessageDialog(this,
            e.getMessage(), "error",
        JOptionPane.ERROR_MESSAGE);
        } catch (IOException e) {
JOptionPane.showMessageDialog(this,
            e.getMessage(), "error",
        JOptionPane.ERROR_MESSAGE);

        } catch (BadLocationException e)
                                {
JOptionPane.showMessageDialog(this,
            e.getMessage(), "error",
        JOptionPane.ERROR_MESSAGE);
        }
        ....
    }
```

La prima cosa da premettere è quella che la comunicazione criptata attraverso tunnel SSH non verrà trattata nel presente articolo, bensì costituirà l'oggetto principale del prossimo. Per cui, facendo riferimento alla **figura 5**, potete completamente ignorare tutte le opzioni riguardanti la connessione SSH. Naturalmente anche nel codice verranno quindi

omesse quelle parti in cui si fa riferimento a tale opzione; in altre parole riportiamo il codice che viene eseguito quando la checkbox per l'abilitazione del tunnel SSH.
Per quanto riguarda l'invio del messaggio, il codice è veramente semplice:

- Se ancora non lo si è fatto si istanzia un oggetto del tipo *IMClient* a cui viene passato l'indirizzo IP ed

Fig. 1: Finestra di login che appare all'avvio dell'applicazione

il numero di porta dell'utente con cui vuole comunicare recuperandoli dalle relative textbox.
- Leggere il messaggio e passarlo all'invocazione del metodo send
- Inserire il messaggio anche nel pannello dove viene riportato tutto le storico della chat.

Il metodo send della classe *IMClient* è così implementato:

```
public void send(String text) throws
        UnknownHostException, IOException {
    PrintWriter out = null;
    Socket s = null;
    int portToConnect;
    String hostToConnect;
    ......
        portToConnect = port;
        hostToConnect = host;
    }
    try {
        s = new
        Socket(hostToConnect,portToConnect);
        out = new
        PrintWriter(s.getOutputStream(),true);
        out.println("[" + user +
                "] " + text);
    } catch (UnknownHostException
                e) {
        throw e;
    } catch (IOException e) {
        throw e;
    } finally {
        if (out != null)
                out.close();
```

```
            if (s != null) s.close();
        }
```

Come ormai abbiamo già più volte visto, la comunicazione avviene attraverso gli stream della coppia client / server.
Andiamo quindi per ordine e vediamo come il lato server, ossia la classe IMServer, integri questa volta al proprio interno anche la classe che si occupa della gestione del protocollo di comunicazione.

UN SERVER ED UN PROTOCOLLO DI COMUNICAZIONE

Quando si utilizza una classe che si fa carico della gestione logica della comunicazione si ottiene come conseguenza che la complessità della classe che funge da server vero e proprio si riduce drasticamente. Infatti ecco come si presenta la classe server:

```
public class IMServer implements Runnable {
        ...........
public IMServer(int port,
        SynchronousQueue<Message> queue) throws
                IOException
    {
        PORT = port;
        this.queue = queue;
        server = new
                ServerSocket(PORT);
        protocol = new IMProtocol();
    }
```

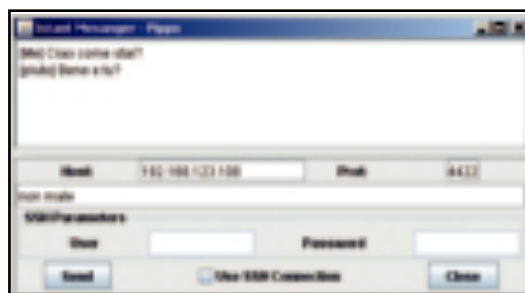



Fig. 2: Sessione di chat tra due utenti

Vedremo tra pochissimo a cosa serve la coda passata al costruttore, per ora diamo un'occhiata a quali azioni compie il server appena viene lanciato:

```
    while (loop) {
            try {
                    Socket socket
                    = server.accept();
                    Message m =
                    protocol.process(socket);
queue.add(m);
```

```
                    if(socket !=
                              null)
socket.close();
            } catch (IOException e)
                                  {
System.err.println(e.getMessage());
            }
        }
```

Tutto qui! Infatti sarà l'oggetto *protocol* a trattare le informazioni ricevute tramite il socket del client ed a restituire un altro oggetto di tipo Message. Quest'ultimo viene poi accodato nell'oggetto *queue* passatogli dalla GUI. È, a mio avviso, importante sottolineare questo aspetto: introducendo la classe *IMProtocol* non ci siamo limitati a spostare il codice da una classe all'altra, abbiamo bensì disaccoppiato due compiti ben distinti. Infatti una cosa è tenere aperta una porta sullo stack TCP/IP ed instradare le comunicazioni su di essa, altro è gestire la logica con cui la comunicazione avviene. In questo modo se si volesse modificare il protocollo di comunicazione basterà implementare nuovamente il metodo process lasciando inalterato tutto il resto.
Infine sarà poi compito del metodo di spooling quello di andare a prelevare i messaggi dalla coda e scriverli sul pannello della chat).
Procediamo con ordine analizzando prima di tutto la classe protocol:

```
public class IMProtocol {
......
    public Message process(Socket sender) {
        BufferedReader reader = null;
        try {
            reader = new
                        BufferedReader(new
    InputStreamReader(sender.getInputStream()));
            StringBuilder sb = new  StringBuilder();
            String line;
            while((line = reader.readLine()) != null)
sb.append(line);
            Message m =
                createMessage(sb.toString());
m.setSenderIP(sender.getRemoteSocketAddress()
                        .toString());
            return m;
        } catch (IOException e) {
            .....
        } finally
        {
            if(reader != null)
                try {
reader.close();
                } catch
                (IOException e) {
e.printStackTrace();
                }
```

```
            }
        }
```

La logica di tale metodo è abbastanza semplice, viene creato un oggetto di tipo Message e si valorizzano i campi da esso previsti, in particolare gran parte di ciò viene nel seguente metodo:

```
private Message createMessage(String text){
        Message m = new Message();
        m.setText(text);
        m.setReceptionDate(new
    java.util.Date(System.currentTimeMillis()));
        return m;
    }
```

LO SPOOLING DELLA CODA

È stato già visto che il server è un thread demone rispetto all'applicazione principale, quindi si è presentata la necessità di far comunicare questi ultimi due. L'oggetto più indicato per assolvere tale compito è sicuramente *SynchronousQueue*. Infatti, riportando quanto scritto nella documentazione ufficiale della SUN per le API Java, una *SynchronousQueue* è una *BlockingQueue* nella quale ogni invocazione del metodo put deve aspettare l'invocazione del metodo take e viceversa.
Come avrete sicuramente notato il server, per inserire un messaggio sulla coda, fa uso del metodo add anziché di put. Il motivo di tale scelta è molto semplice: non vogliamo assolutamente che il server si blocchi su questo metodo finché dall'altra parte qualcuno (nella fattispecie lo spooler della GUI) invochi il metodo take.
Nella situazione diametralmente opposta si trova invece lo spooler. Infatti lo spooler non deve fare niente, ossia deve attendere, finché la coda non contiene almeno un messaggio. Una volta preso, lo dovrà poi scrivere sul pannello della chat. Anche in questo caso risulta naturalmente necessario che lo spooler sia anche esso un demone figlio del thread principale della GUI. Vediamo quindi come quanto detto finora sia stato implementato:

```
    public IMGUI(String name) {
        ......
        spooler = new Thread(new
                        Runnable(){
            public void run() {
                spool();
            }
        });
        spooler.setDaemon(true);
        spooler.start();
        .......
```

Al momento della creazione della GUI viene creato il demone di spooling e lanciato. Essendo un demone il suo compito non è altro quello di effettuare un loop infinito per controllare la coda:

```
private void spool() {
        while (true) {
            try {
                Message m = queue.take();
                int length =
                jEditorPaneChat.getDocument().getLength();

                String text = m.getText();
                SimpleAttributeSet att = new SimpleAttributeSet();
StyleConstants.setItalic(att, true);
jEditorPaneChat.getDocument().insertString(length,
                text+"\n", att );
            } catch (Exception e) {
JOptionPane.showMessageDialog(this,
                e.getMessage(), "error",
                JOptionPane.ERROR_MESSAGE);
            }
        }
    }
```

L'invocazione del metodo take farà sì che il loop determinato dal ciclo while(true) si blocchi in questo punto ed attenda finché un messaggio non viene inserito nella coda. Una volta ricevuto il messaggio ne formatterà il contenuto e lo inserirà come ultima riga nello storico della chat.

CONCLUSIONI

Ora, per concludere, possiamo quindi porci qualche domanda interessante:

1 – Tra l'utente “Pippo” e l'utente “Pluto” cosa c'è in mezzo? Ossia siamo sicuri che l'applicazione funzioni allo stesso modo indipendentemente dal numero di nodi che si frappongono tra i due?

2 – Se qualcuno si mettesse a fare sniffing sulla porta da noi scelta, quanto riuscirebbe a capire di quello che stiamo scrivendo?

3 – Un port scannig sul pc nel quale stiamo eseguendo l'applicazione che conseguenze potrebbe avere e come ci potremmo nascondere da una simile indagine maliziosa?

A tutto questo daremo risposta nella prossima puntata.

Andrea Galeazzi

REALIZZARE L'INTERFACCIA

0Non rientra negli scopi dell'articolo la spiegazione dello sviluppo di un'interfaccia grafica con la libreria swing. A titolo di cronaca per realizzare questa applicazione mi sono avvalso dell'IDE Eclipse e del relativo plugin Jigloo.

RIPROGRAMMARE IL CODICE CON STILE

HAI SCRITTO IL TUO FANTASTICO SOFTWARE. ORA TI ACCORGI CHE AVRESTI POTUTO USARE NOMI PIÙ SIGNIFICATIVI. COME MIGLIORARE IL TUO SOFTWARE SENZA DOVER CAMBIARE MANUALMENTE TUTTI I RIFERIMENTI? A QUESTO È A MOLTO ALTRO CI PENSA IL REFACTORING!

Il **Refactoring** può essere definito come una qualsiasi modifica al codice sorgente di un programma per computer che ne migliora la leggibilità o ne semplifica la struttura senza però modificarne il funzionamento e quindi il risultato. Da questa definizione traiamo immediatamente il significato di Refactoring e quanto quindi questo processo sia importante per migliorare il nostro lavoro. Non esistono delle vere e proprie liste di regole per il refactoring scolpite nella roccia ma esistono una serie di buone pratiche di programmazione che se seguite permettono di produrre codice di qualità facilmente estendibile, leggibile e manutenibile. Quindi come spesso avviene in rete, nel tempo alcune di queste pratiche sono entrate nell'uso comune ed essendosi rivelate utili nonché sufficientemente testate sono state formalizzate sotto forma di refactoring nel libro: *Refactoring: Improving the Design of Existing Code* di *Martin Fowler*. Microsoft, dal canto suo, è ben conscia dell'importanza del processo di refactoring del codice e di conseguenza con il rilascio della versione *2005* di *Visual Studio* ha introdotto tutta una serie di funzionalità per aiutare il programmatore in tal senso. E non solo per il codice di programmazione, infatti la versione *for Database Professionals* di *Visual Studio Team Edition* comprende anche una funzionalità di refactoring specifica per i database che consente di rinominare una colonna di una tabella aggiornando di conseguenza anche tutti i riferimenti alla stessa ad esempio nelle stored procedure.
Prima dell'arrivo dei tool, tra cui *Visual Studio 2005*, il lavoro di refactoring del codice era abbastanza lungo e laborioso e richiedeva un pesante intervento manuale nel codice. Inoltre l'intervento manuale facilmente poteva portare all'introduzione di errori senza contare la consistente perdita di tempo. Grazie ai tool di refactoring ed in particolare a *Visual Studio 2005*, oggi è possibile effettuare questo processo con una semplicità mai vista prima. Con pochi click del mouse è possibile modificare una intera soluzione senza che questo comporti l'introduzione di errori e senza perdita di tempo. Vediamo quindi quali sono queste funzioni che *Visual Studio 2005* ci mette a disposizione e quando utilizzarle.

Conoscenze richieste
Visual Studio 2005 e C#

Software
Windows XP Service Pack 2, Windows Server 2003 o Windows Vista – .NET Framework 2.0, Visual Studio 2005 (standard o professional).

Impegno

Tempo di realizzazione

LE TECNICHE DI REFACTORING

Vediamo quindi dettagliatamente quali sono le tecniche di refactoring presenti in *Visual Studio 2005* ed alcuni esempi che ci permetteranno di capire quando è opportuno applicarle. Innanzi tutto vediamo un elenco sintetico dei refactoring insieme ad una loro breve descrizione:

- **Extract Method**
  Permette di creare un nuovo metodo a partire dal codice selezionato
- **Encapsulate Field**
  Converte un campo pubblico in un campo privato esposto da una proprietà
- **Extract Interface**
  Crea una interfaccia sulla base di un gruppo di tipi esistenti
- **Reorder Parameters**
  Consente di riordinare i parametri di un metodo
- **Remove Parameters**
  Consente di rimuovere un parametro da un metodo
- **Rename**
  Permette di rinominare un elemento (un metodo, una classe, una proprietà, …) all'interno dell'intero progetto
- **Promote Locale Variable to Parameter**
  Converte una variabile locale in un parametro del metodo in cui è contenuta
- **Generate Method Stub**
  Crea un metodo a partire da una sua chiamata.

Tutte queste funzioni, tranne l'ultima, in *Visual Studio 2005* sono accessibili sia tramite l'apposito menu *Refactor* che tramite il menu contestuale visualizzato cliccando con il tasto destro del mouse

Figura 1: Menu Refactor

Figura 2: Menu contestuale dei refactor

all'interno del codice.
L'unica eccezione è il comando Generate Method Stub che invece appare sempre nel menu contestuale ma fuori dal gruppo di voci Refactor oppure all'interno del codice tramite la funzionalità di Automatic Code Generation dell'IntelliSense direttamente sulla chiamata al metodo

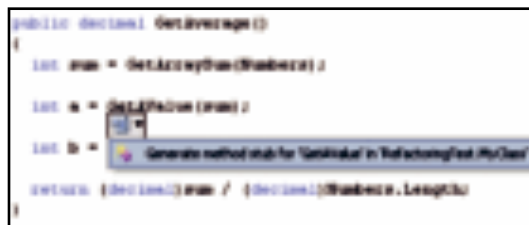



Figura 3: Comando Generate Method Stub

EXTRACT METHOD

La tecnica di refactor *Extract Method* ci permette di estrapolare un nuovo metodo da una porzione di codice esistente. Accade sovente durante lo sviluppo di scrivere porzioni di codice che in qualche modo possono essere riutilizzate in altre parti del progetto. Sarebbe quindi scomodo e ridondante ripetere diverse volte le stesse righe di codice, senza contare il fatto che in caso di modifica dovremmo necessariamente modificare tutti i punti nel progetto in cui queste righe di codice sono state utilizzate. Ecco che quindi ci viene in aiuto questa tecnica di refactoring. Vediamo il seguente semplice esempio:

```
public class MyClass
{
    public MyClass()
    { }

    public decimal GetAverage( int[] numbers)
    {
      int sum = 0;

      foreach (int number in numbers)
      {
        sum += number;
      }

      return (decimal)sum / (decimal)numbers.Length;
    }
}
```

Questa classe contiene il metodo *GetAverage* che preso un array di valori *int* ne calcola la media aritmetica. Come possiamo vedere i momenti del calcolo sono due: il primo quando calcoliamo la somma di tutti i valori dell'array, il secondo quando dividiamo questa somma per il numero di elementi. E' evidente che il calcolo della somma degli elementi è una operazione di cui possiamo facilmente aver bisogno in altre parti della nostra applicazione, è quindi opportuno separare questo codice dal metodo *GetAverage* affinchè possa vivere di vita propria ed essere riutilizzato. Per fare questo utilizziamo allora il comando *Extract Method*, selezionando le righe di codice relative al calcolo della somma e cliccando con il tasto destro del mouse sulla selezione:

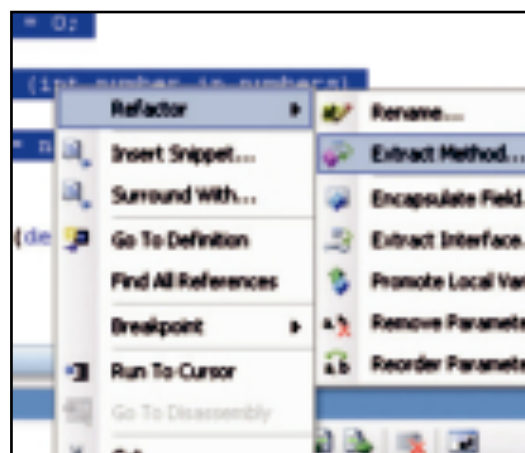



Figura 4: Voce Extract Method del menu contestuale

Selezioniamo quindi *Extract Method*. Vediamo apparire una finestra di dialogo nella quale dobbiamo specificare solo il nome del nuovo metodo da creare, nel nostro caso *GetArraySum*:

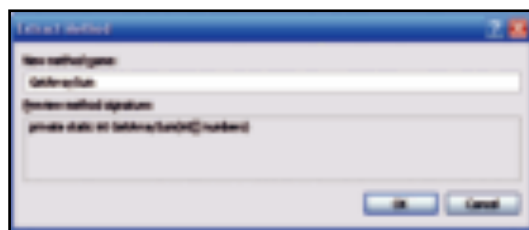



Figura 5: Dialog del comando Extract Method

Nella stessa finestra possiamo vedere anche un'anteprima di come apparirà la firma del metodo. Premuto quindi il tasto *OK* il nostro codice viene modificato in questo modo:

```
public class MyClass
{
    public MyClass()
    { }
    public decimal GetAverage( int[] numbers)
    {
      int sum = GetArraySum(numbers);
      return (decimal)sum / (decimal)numbers.Length;
    }
    private static int GetArraySum(int[] numbers)
    {
      int sum = 0;

      foreach (int number in numbers)
      {
        sum += number;
      }
      return sum;
    }
}
```

Come possiamo vedere *Visual Studio* ha fatto un ottimo lavoro. Innanzi tutto ha creato un nuovo metodo *GetArraySum* contenente le righe di codice da noi selezionate precedentemente, poi ha creato un parametro del metodo corrispondente all'array di input su cui calcolare la somma ed ha inserito come valore di ritorno del metodo la variabile *sum*. Fatto questo ha sostituito tutto il codice selezionato con la chiamata al nuovo metodo passandogli come parametro proprio l'array di numeri da sommare. A questo punto possiamo tranquillamente togliere il metodo *GetArraySum* dalla nostra classe e posizionarlo magari in una classe *Helper* da cui potrà essere richiamato molteplici volte da altre classi del progetto.

ALTRE INFORMAZIONI SUL REFACTORING

Vi sono diverse tecniche di refactoring e numerose sono le discussioni che ruotano attorno a questo tema. All'indirizzo: www.refactoring.com possiamo trovare il sito di supporto al testo di Martin Fowler citato nell'introduzione. Qui troviamo diverse risorse che parlano di refactoring nonché numerosi tool da scaricare per sottoporre a refactoring il nostro codice sorgente. In particolare degna di nota è la pagina: http://www.refactoring.com/catalog/index.html che elenca un nutrito numero di tecniche di refactoring. Informazioni di carattere più generale sul refactoring sono anche disponibili su Wikipedia.org all'indirizzo: http://en.wikipedia.org/wiki/Refactoring

ENCAPSULATE FIELD

Encapsulate Field svolge anch'esso un compito molto utile, ovvero permette di incapsulare un campo pubblico all'interno di una proprietà utilizzando un campo privato come variabile di appoggio. Tutti sappiamo che non è cosa buona e giusta definire un campo pubblico all'interno di una classe perché questo viola il principio dell'encapsulation e non ci permette di avere il controllo dei valori assegnati al campo. Può accadere però per nostra sfortuna di dover ricevere in consegna del codice da modificare sviluppato da qualcuno che non ha rispettato queste regole di buona programmazione e quindi ci tocca rimediare. *Visual Studio* ci viene in aiuto proprio con questa tecnica di refactoring. È sufficiente cliccare con il tasto destro del mouse sul campo pubblico che vogliamo incapsulare e scegliere solo tra alcune semplici opzioni all'interno della finestra di dialogo che ci appare:

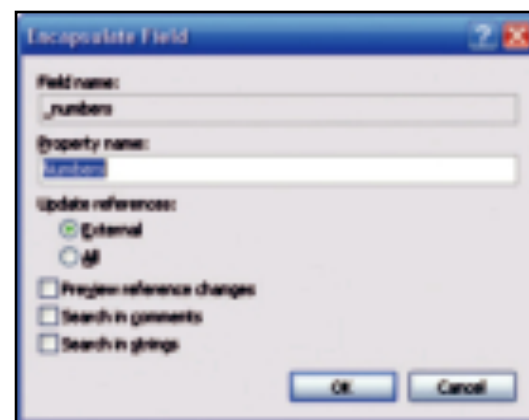



Figura 6: Dialog del comando Encapsulate Field

Dobbiamo nell'ordine selezionare il nome da dare alla proprietà, il modo in cui deve essere effettuato l'aggiornamento dei riferimenti, se visualizzare una preview prima di apportare le modifiche, se effettuare le sostituzioni anche nei commenti e nelle stringhe di testo. La modalità di aggiornamento **External** permette di aggiornare tutti i riferimenti del campo pubblico nel progetto eccetto quelli della classe stessa in cui il campo è contenuto, mentre la modalità **All** aggiorna tutti i riferimenti

al campo, anche quelli presenti nella stessa classe in cui è definito.
Tornando al nostro esempio della classe che calcola la media, ipotizziamo di averla modificata in modo tale da contenere un campo che permette di specificare l'array sul quale calcolare la media piuttosto che passarlo come parametro dal relativo metodo:

```
public class MyClass
{
    public int[] _numbers;

    public MyClass()
    { }
    public decimal GetAverage()
    {
      int sum = GetArraySum(_numbers);

      return (decimal)sum /
                        (decimal)_numbers.Length;
    }

    ...
}
```

Applichiamo quindi l'*Encapsulate Field* come spiegato sopra (specificando come metodo di aggiornamento *All*) e premiamo *OK*. Vedremo il codice diventare:

```
public class MyClass
{
    private int[] _numbers;

    public int[] Numbers
    {
      get { return _numbers; }
      set { _numbers = value; }
    }

    public MyClass()
    { }

    public decimal GetAverage()
    {
      int sum = GetArraySum(Numbers);

      return (decimal)sum /
                        (decimal)Numbers.Length;
     }

    ...
}
```

Le modifiche apportate sono: *private* al posto di *public* sul campo *_numbers*, la creazione della nuova proprietà *Numbers* che espone il campo privato *_numbers* e la sostituzione nel codice di tutti i riferimenti a *_numbers* in riferimenti alla nuova proprietà *Numbers*. Tutto questo con un paio di click senza contare eventuali altri riferimenti al campo *_numbers* utilizzati nel progetto che sono stati ora automaticamente aggiornati verso la proprietà che lo ha inglobato.

EXTRACT INTERFACE

Questo comando estrae una interfaccia da una classe. Conosciamo tutti i pregi delle interfacce e quanto queste possano risultare utili nello sviluppo software. Ecco quindi che nel momento in cui ci accorgiamo che una particolare classe presenta metodi, proprietà od eventi che potrebbero tranquillamente essere fatti propri da altre classi, automaticamente pensiamo di creare una interfaccia da cui far derivare la classe. Grazie ad *Extract Interface* possiamo fare questo facilmente. È sufficiente posizionarsi con il cursore all'interno di una classe e cliccare in un punto vuoto qualsiasi con il tasto destro del mouse. Scegliamo dal menu *Refactor* la voce *Extract Interface* e nell'ordine specifichiamo: il nome da dare all'interfaccia, il nome del file su disco che la conterrà, i metodi da includere nell'interfaccia. Riprendendo la nostra classe per il calcolo delle medie aritmetiche, è evidente che

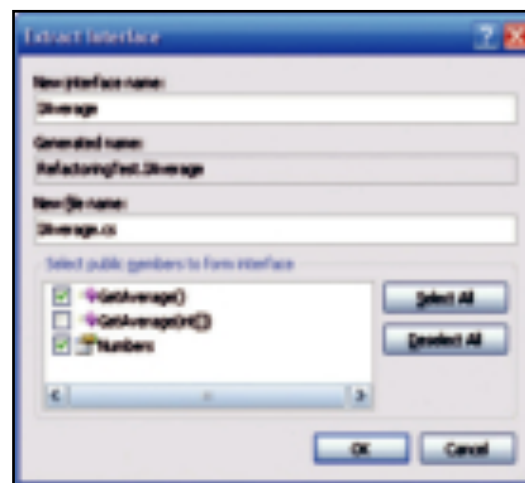



Figura 7: Dialog del comando Extract Interface

NON SOLO VISUAL STUDIO

Ecco un elenco di altri ottimi tool di refactoring:

- **C# Refactory (http://www.xtreme-simplicity.net/csharpefactory.html)**
- **ReSharper (http://www.jetbrains.com/resharper/)**
- **devAdvantage (http://www.anticipatingminds.com/content/products/devadvantage/devadvantage.aspx)**
- **Refactor! for Visual Basic 2008 (http://msdn2.microsoft.com/en-us/vbasic/bb693327.aspx)**
- **JustCode! (http://www.omnicore.com/en/justcode.htm)**

possiamo individuare dei metodi e proprietà generici che possono essere utilizzati anche per il calcolo di medie diverse da quella aritmetica (geometrica, quadratica, ecc.). Generiamo quindi una interfaccia che si chiami *IAverage* e che includa la proprietà *Numbers* ed il metodo *GetAverage()*.
Dopo la pressione del tasto *OK* viene generato il file *IAverage.cs* contenente la seguente interfaccia:

```
using System;
namespace RefactoringTest
{
  interface IAverage
  {
    decimal GetAverage();
    int[] Numbers { get; set; }
  }
}
```

E la nostra classe viene fatta ereditare da *IAverage*:

```
public class MyClass : RefactoringTest.IAverage
{
    private int[] _numbers;
    …
```

REORDER PARAMETERS

Consente di riordinare i parametri di un metodo. Questa operazione può sembrare a prima vista inutile ma vi sono casi in cui invece si rende quasi necessaria. Ipotizziamo di aver creato una serie di metodi in overload:

```
public decimal GetAverage( int[] numbers)
{ … }

public decimal GetAverage( int[] numbers, Averages
                                               avg)
{ … }

public decimal GetAverage( int[] numbers, int a,
                                      Averages avg)
{ … }
```

Notiamo subito che l'ultimo overload presenta il parametro "*a*" prima del parametro "*avg*" ma sarebbe più corretto che quest'ultimo mantenesse la stessa posizione rispetto agli overload precedenti (quindi la seconda posizione) e che il nuovo parametro "*a*" andasse in coda agli altri. Se ci rendiamo conto di questo errore dopo che oramai l'overload errato è stato già utilizzato diverse volte nel codice diventa un bel problema modificarlo. *Reorder Parameters* ci permette invece di effettuare questo cambiamento con semplicità. Dobbiamo soltanto selezionare il metodo i cui parametri desideriamo riordinare e dal consueto menu contestuale che appare cliccando con il tasto destro del mouse, selezionare dal sotto menu *Refactor* la voce *Reorder Parameters*.

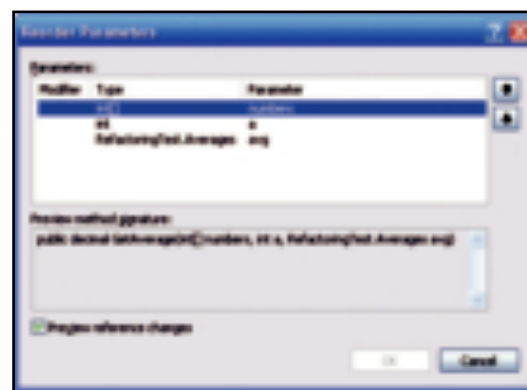



Figura 8: Dialog del comando Reorder Parameters

In questa finestra scegliamo l'ordine corretto dei parametri (spostiamo il parametro *a* in coda agli altri) agendo su di essi con i tasti freccia sulla destra e poi premiamo il tasto *OK* dopo aver selezionato magari l'opzione per visualizzare una preview delle modifiche prima di confermarle.
Il parametro è stato quindi correttamente spostato dopo tutti gli altri e altresì qualsiasi riferimento a questo metodo all'interno del progetto e degli altri collegati, è stato aggiornato per rispecchiare la nuova disposizione dei parametri.

MODIFICHE MULTIPROGETTO

Le modifiche apportate dalle funzioni di refactoring non sono applicate solo nel progetto corrente ma sono estese anche a tutti i progetti correlati. Ad esempio se si sta sviluppando un Windows Form che utilizza un progetto Class Library ed effettuiamo una modifica di refactoring (ad esempio una Rename di un metodo) all'interno della Class Library, saranno aggiornati anche tutti i riferimenti del progetto Windows Form verso quel metodo o tipo modificato.

REMOVE PARAMETERS

Come dice il nome stesso, questo comando rimuove un parametro da un metodo aggiornando di conseguenza tutte le chiamate al metodo stesso. Selezioniamo anche in questo caso il metodo su cui agire e dal menu contestuale dei *Refactor* scegliamo la voce *Remove Parameters*. Viene visualizzata una finestra di dialogo molto semplice che chiede di specificare i parametri da rimuovere o eventualmente ripristinare (nello stesso momento, non dopo che la modifica è stata apportata).
Come anche il messaggio di alert dice nella fine-

Figura 9: Dialog del comando Remove Parameters

stra di dialogo, bisogna fare attenzione nel rimuovere parametri che sono stati utilizzati all'interno del metodo stesso perché eliminandoli si potrebbero venire a creare situazioni di errore dovute al fatto che il parametro viene utilizzato senza essere stato dichiarato.

RENAME

Il comando *Rename* permette di rinominare un metodo, un namespace, un parametro od un tipo, apportando la modifica all'intero progetto in cui è contenuto l'elemento modificato. Naturalmente questo comando di refactoring non esegue una semplice *find e replace* ma considera il particolare elemento selezionato e quindi apporta le dovute sostituzioni. Per questo motivo se ad esempio abbiamo un metodo che si chiama *Connection* e da qualche altra parte una proprietà che si chiama anch'essa *Connection* (sebbene questo è preferibile evitarlo), rinominando con il comando *Rename* il metodo, non verrà rinominata anche la proprietà ne tanto meno tutti i riferimenti ad essa. Il comando *Rename* richiede solo di specificare il nuovo nome da dare all'elemento selezionato e, come per altri comandi di refactoring, richiede di specificare se apportare le modifiche anche nei commenti e nelle stringhe testuali, oltre naturalmente alla possibilità di visualizzare una finestra di preview prima di procedere con l'operazione. Nel caso specifico della ridenominazione di un metodo, possiamo anche scegliere se rinominare gli overload dello stesso oppure no.

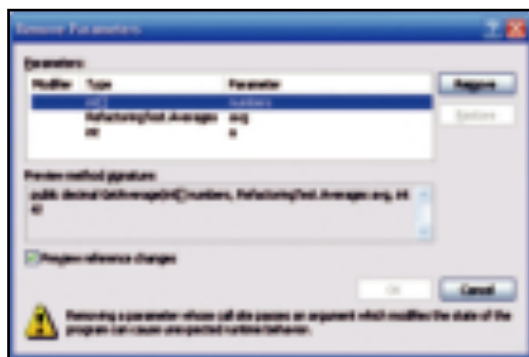



Figura 10: Dialog del comando Rename

PROMOTE LOCALE VARIABLE TO PARAMETER

Può accadere che dopo aver scritto un metodo ci si accorga che alcune variabili locali in esso definite sarebbe preferibile richiederle come parametri nella chiamata del metodo piuttosto che valorizzarle all'interno dello stesso riducendone in questo modo la flessibilità. Il comando *Promote Locale Variable to Parameter* "promuove" la variabile locale selezionata a parametro del metodo, spostandone quindi semplicemente la definizione. Poniamo il caso di avere un metodo che salva una riga in un file di log. In prima battuta scriviamo:

```
public class LogManager
{
    public static void WriteLog(string logText)
    {
        string logFilePath = @"C:\Logs\MyLog.txt";
        using (StreamWriter stm = File.AppendText(
                                                logFilePath))
        {
            TextWriter tw =
                        TextWriter.Synchronized(stm);
            tw.WriteLine(logText);
            tw.Flush();
            tw.Close();
        }
    }
}
```

Poi ci accorgiamo che sarebbe più comodo se il nome del file di log fosse specificato nella chiamata stessa del metodo *WriteLog* in quanto in questo modo possiamo scrivere in file di log differenti a seconda del punto del progetto in cui ri-

RESOURCE REFACTORING TOOL

Si tratta di un add-in per Visual Studio 2005 che aggiunge ai già citati comandi di refactoring, il comando Extract to Resource. In pratica eseguendo questo comando su una qualsiasi stringa di testo specificata nel nostro codice, possiamo estrapolarla ed inserirla velocemente in un file di risorse. Al suo posto naturalmente sarà inserito il codice opportuno per richiamare la stringa stessa. La pagina principale del progetto ospitata su CodePlex è questa: http://www.codeplex.com/ResourceRefactoring

chiamiamo il metodo. Promuoviamo quindi la variabile *logFilePath* a parametro cliccando con il tasto destro del mouse sul suo nome nella definizione e dal gruppo *Refactor* scegliendo *Rename*. Immediatamente vedremo la definizione della variabile sparire dal corpo del metodo per riapparire come parametro nella firma dello stesso:

```
public static void WriteLog(string logText, string
                                                logFilePath)
{
      using (StreamWriter stm = File.AppendText(
                                              logFilePath))
      {
        TextWriter tw =
TextWriter.Synchronized(stm);
        tw.WriteLine(logText);
        tw.Flush();
        tw.Close();
      }
    }
}
```

Così facendo abbiamo reso più flessibile il nostro codice e più facilmente riutilizzabile.

GENERATE METHOD STUB

Per ultimo vediamo questo altrettanto comodo comando che ci appare in un menu dell'*IntelliSense* nel momento in cui scriviamo una chiamata di un metodo che non esiste ancora nel namespace corrente. *Visual Studio* automaticamente individua l'assenza del metodo nella classe e quindi ci mostra tramite l'*IntelliSense* una sottolineatura breve all'inizio del nome del metodo, passando sulla quale ci viene visualizzata una voce di menu come quella in **figura 3**. Selezionando questa voce di menu verrà creato lo scheletro di un metodo che avrà la stessa firma di quello da noi scritto. Automaticamente verranno riconosciuti i tipi dei parametri e di conseguenza saranno incluse nella firma le corrette definizioni. Se ad esempio senza aver ancora definito il metodo *GetBookDescription*, scriviamo il seguente codice:

```
string title = "";
string author = "";
decimal price = 0;
string description = GetBookDescription
( title, author, price);
```

Selezionando *Generate Method Stub* dal menu che verrà visualizzato, *Visual Studio* genererà automaticamente il codice seguente:

```
private string GetBookDescription
(string title, string author, decimal price)

{
     throw new Exception
("The method or operation
                               is not implemented.");

 }
```

Ovvero lo scheletro del metodo *GetBookDescription* con i tre parametri già opportunamente tipizzati così come il valore di ritorno. Non ci resta quindi altro da fare che riempire il metodo con il codice opportuno.

CONCLUSIONI

Come abbiamo visto i comandi di refactoring che *Visual Studio 2005* ci offre sono molto potenti ed anche facili da usare. Naturalmente questi comandi non sono tutti quelli possibili ed altri tool magari più specifici possono sicuramente offrire qualcosa di più. Il concetto più importante però da capire è che produrre codice ben scritto, leggibile e perché no, elegante, non deve essere visto come una perdita di tempo o come una inutile sofisticazione perché i vantaggi concreti sui tempi di sviluppo e sulle prestazioni, nonché sulla manutenibilità non tarderanno certo ad arrivare quando sviluppare seguendo queste regole sarà diventata un'abitudine. Non dimentichiamoci infine che rivedere il codice per migliorarlo è un processo continuo che va pianificato come parte integrante dello sviluppo software e come tale valorizzato ed assimilato.

Gianni Malanga

COSA È L'AUTOMATIC CODE GENERATION

Il menu dell'IntelliSense accennato nell'articolo relativamente al comando Generate Method Stub, è uno dei menu contestuali che permettono di eseguire una serie di azioni specifiche per un particolare elemento del codice che costituiscono la cosiddetta funzionalità di Automatic Code Generation. Questa funzionalità del più generico sistema IntelliSense, permette di inserire del codice li dove è prevedibile che venga inserito. Ecco che quindi abbiamo la possibilità di inserire automaticamente una using al posto del nome completo di un namespace, creare un metodo sulla base di una sua chiamata (Generate Method Stub), creare associazioni ai gestori di evento, implementare le classi base di tipo abstract o implementare i membri di una interfaccia.

MICROSOFT ENTITY FRAMEWORK

LA MAGGIOR PARTE DELLE APPLICAZIONI FA USO DI DATI. LA LORO GESTIONE È SEMPRE STATO UNO DEI PROBLEMI FONDAMENTALI DELLA PROGRAMMAZIONE. VEDIAMO COME TUTTO CI VIENE RESO PIÙ SEMPLICE ATTRAVERSO I NUOVI STRUMENTI DI MICROSOFT

Conoscenze richieste
C#, Conoscenze base del .NET Framework 3.0 e di LINQ to Entities

Software
Windows XP Service Pack 2, Windows Server 2003 o Windows Vista – .NET Framework 3.5 Beta 2, Visual Studio 2008 (Orcas) Beta 2 o Visual Studio 2005

I dati e la loro gestione rappresentano uno dei punti fondamentali di qualsiasi progetto software ed anche uno degli aspetti considerati più noiosi nello sviluppo di un'applicazione. Gli sviluppatori devono utilizzare i database ma non vedono di buon occhio il codice ripetitivo da scrivere per persistere i diversi oggetti di volta in volta necessari.
A questo scopo sono nati nel tempo diversi strumenti con l'intento di semplificare questo aspetto dello sviluppo software. Strumenti come i *data abstraction layer*, i *persistence layer* o gli *object relational mapping tool* tra cui ricordiamo a titolo di esempio il famoso *NHibernate*, vengono in aiuto dello sviluppatore per accelerare lo sviluppo e rendere più ottimizzato il codice. Il problema è che ognuno di questi strumenti ha i suoi pregi e difetti e nessuno di essi può definirsi realmente la soluzione migliore per la gestione della persistenza dei dati.
La scelta dello strumento più adatto da utilizzare dipende sempre dal tipo di applicazione e dai requisiti della stessa. Alle volte quindi gli stessi sviluppatori non trovando lo strumento adatto alla situazione specifica, preferiscono orientarsi verso soluzioni auto prodotte che sono le uniche in grado di adattarsi perfettamente al contesto applicativo anche se richiedono uno sforzo implementativo di certo non indifferente.
In questo scenario, quando Microsoft annunciò che avrebbe rilasciato una nuova tecnologia denominata **Entity Framework** i più appresero la notizia con un misto di curiosità e scetticismo.
La prima naturale reazione fu un confronto con le tecnologie esistenti, come il già citato NHibernate. Questo però è stato un errore perché in realtà l'*Entity Framework* di Microsoft è una soluzione totalmente diversa dagli altri prodotti e nasce con finalità totalmente diverse. Ma quindi cos'è l'*Entity Framework*?

MICROSOFT ENTITY FRAMEWORK

L'*Entity Framework* nasce con l'intento di unificare il modo in cui i diversi componenti di un team di sviluppo software (sviluppatori, analisti, database administrator) concepiscono la gestione della base dati, fornendo un linguaggio comune per raggiungere i medesimi obiettivi. Il cuore dell'*Entity Framework* è l'*Entity Data Model* (*EDM*), il nucleo che racchiude la sintassi comune utilizzata nei diversi ambiti applicativi. L'*EDM* definisce un linguaggio con il quale poter descrivere i dati senza doversi preoccupare di descrivere anche il contenitore degli stessi, ovvero il database. Subito sopra questo nucleo comune troviamo un insieme di servizi preposti alla manipolazione dei dati.
Ricapitoliamo quindi cosa *non* è l'*Entity Framework*:

- L'EF non è solo un ORM (*Object Relational Mapping tool*)
- L'EF non è solo un engine per la persistenza dei dati
- L'EF non è solo un generatore di codice

L'*Entity Framework* è invece un set di servizi incentrati sui dati, quindi un *data service framework*. Tra questi servizi troviamo quindi anche la persistenza dei dati, anche servizi per la gestione di oggetti o per la gestione di viste, di query, e così via. Possiamo quindi definirlo come un framework che mette insieme un po' tutti gli aspetti relativi alla gestione dei dati all'interno di un'applicazione software.
Come dice il suo stesso nome, l'unità di misura dei dati è l'**Entità**. Ma cos'è una entità? Per l'*Entity Framework* una entità è un qualcosa che ha le seguenti caratteristiche:

- È univocamente identificata tramite informazioni contenute in se stessa

- Contiene dati di tipi definiti e specifici
- Ha proprietà che memorizzano dati scalari
- Ha proprietà che memorizzano le relazioni con altre entità
- Non ha metodi (una entità deve descrivere, non eseguire)

La struttura dell'*Entity Framework* vede alla base, sotto i noti provider di *ADO.NET*, la base dati intesa come contenitore di informazioni e non come database in quanto sebbene il database sia la base dati privilegiata utilizzata nella stragrande maggioranza dei casi è possibile in ogni caso persistere i dati anche su *file server* o attraverso *Web Service*. Subito sopra i provider *ADO.NET* troviamo l'*EDM* che sostanzialmente è un insieme di metadati che descrivono i dati utilizzati dall'applicazione

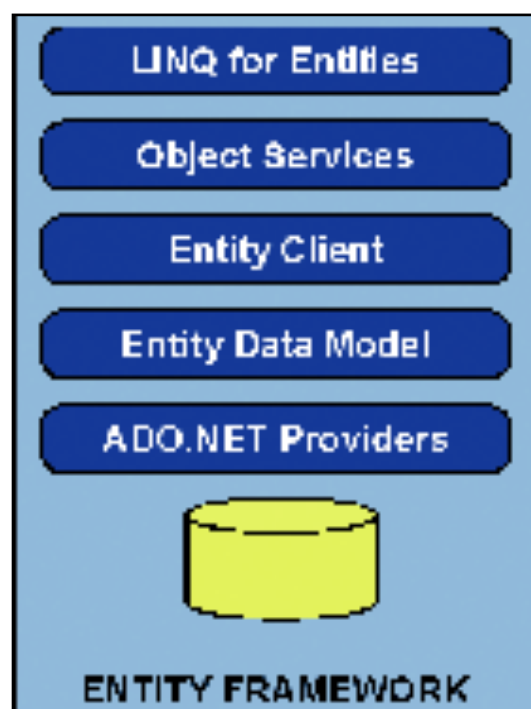



Fig. 1: Architettura dell'Entity Framework

nonché il modo in cui questi sono mappati sulla base dati. Inoltre l'*EDM* fornisce dei meccanismi per separare nettamente il modo in cui le informazioni sono utilizzate dall'applicazione e quindi dall'utente rispetto al modo in cui queste sono memorizzate nella base dati, ad esempio aggregando informazioni provenienti da differenti basi dati in una unica entità

LA TERMINOLOGIA

Con l'*Entity Framework* è stata introdotta una nuova terminologia che è necessario padroneggiare per comprendere con più chiarezza i concetti affrontati. Ecco quindi una tabella riepilogativa dei termini e dei nomi utilizzati quando si parla di *Entity Framework*:

Termine	Descrizione
alias	Attributo degli schemi *CSDL* e *SSDL* utilizzato nello schema stesso come abbreviazione del namespace completo
association	Definizione della relazione tra due entità
association set	Raggruppamento di più *association* appartenenti allo stesso tipo
base type	Tipo base o padre da cui altri tipi dell'Entity Data Model possono ereditare delle proprietà.
complex object	Istanza di un tipo complesso contenuta in un *object context*
complex type	Tipo complesso che rappresenta una proprietà non scalare di una entità e che permette di organizzare le proprietà scalari di una entità per gruppi
container	Gruppo logico di entità e associazioni (*association*)
direction	Riferita alla natura asimmetrica di alcune association, viene specificata negli attributi FromRole e ToRole degli elementi NavigationProperty o ReferentialConstraint.
End	Specifica una entità che partecipa ad una *association*
EntityClient	Data provider ADO.NET indipendente dalla base dati che contiene le classi necessarie per l'accesso ai dati, come: EntityConnection, EntityCommand e EntityDataReader.
entity object	Istanza di un tipo entity che vive all'interno di un *Object Context*
entity set	Contenitore logico di entità di uno stesso tipo
Entity SQL	Dialetto SQL indipendente dalla base dati che dialoga direttamente con lo schema concettuale (CSDL).
facet	Restrizione che specifica i valori ammessi per una particolare proprietà, come ad esempio: MaxLength, Nullable o Precision.
multiplicity	Numero di entità che possono comparire su ciascuna "estremità" di una relazione. Chiamata anche *cardinalità*.
navigation property	Proprietà delle entità che permette di referenziare altre entità attraverso una *association*.
query path	Query che specifica gli oggetti relazionati da recuperare quando viene eseguita una query su una entità
object context	Oggetto che rappresenta il contenitore di tutte le entità definite nello schema concettuale e che contiene la connessione alla base dati sottostante fornendo inoltre diversi servizi tra cui ad esempio il tracciamento delle modifiche ai dati
object query	Query eseguita sull'*Entity Data Model* all'interno del contesto di un *Object Context*. Restituisce i dati sotto forma di oggetti.
relationship	Connessione logica tra entità
role	Nome dato a ciascuna End di ogni association per permetterne l'identificazione
scalar property	Proprietà di una entità mappata su un singolo campo dello *Storage Schema* (*SSDL*)
simple type	Tipo primitivo utilizzato per definire le proprietà delle entità nello schema concettuale

INSTALLAZIONE

Per poter utilizzare l'Entity Framework è necessario aver installato precedentemente sul proprio computer il .NET Framework 3.5

Beta 2 (l'ultima versione disponibile al momento in cui scriviamo). Oppure il nuovo *Visual Studio 2008 (Orcas) Beta 2*. In quest'ultimo caso il *.NET Framework 3.5* è già installato e con esso anche l'*Entity Framework*, quindi in pratica sarà sufficiente installare, se interessati, solo gli *ADO.NET Entity Framework Tools* per essere operativi.

L'installazione del *.NET Framework 3.5 Beta 2* non richiede particolari accorgimenti. Sarà sufficiente avviare il file di setup, accettare i termini della licenza d'uso e far partire il download del Framework dal sito Microsoft. Questa operazione richiederà alcuni minuti a seconda della velocità della propria connessione ad internet. Al termine dell'installazione è consigliato installare gli eventuali aggiornamenti e service pack dopo di che potremo passare all'Entity Framework. Avviamo il file di setup ed anche in questo caso dovremo solo accettare i termini della licenza d'uso, fatto questo, dopo alcuni secondi il setup terminerà e l'Entity Framework sarà pronto per essere utilizzato.

Dopo questa introduzione teorica, nei prossimi paragrafi affronteremo gli aspetti più pratici dell'uso dell'*Entity Framework* tenendo sempre in mente che stiamo trattando un prodotto ancora in fase Beta e quindi con molta probabilità alcune delle cose che diremo potrebbero subire delle variazioni rispetto alla versione definitiva il cui rilascio è comunque previsto a breve.

NOTA

IL BLOG DI ADO.NET

Il blog ufficiale del team Microsoft di sviluppo di ADO.NET è raggiungibile all'indirizzo: http://blogs.msdn.com/adonet/ Qui sono disponibili diverse informazioni ufficiali su ADO.NET ed in particolare sull'Entity Framework. Tutte le future nuove versioni e release verranno comunicate su questo blog prima che su qualunque altro sito.

ENTITY DATA MODEL

L'*Entity Data Model (EDM)* come abbiamo detto è il nucleo dell'*Entity Framework* in quanto questo permette la definizione delle entità, le relazioni che intercorrono tra di esse ed il raggruppamento delle stesse in “viste” logiche.

La definizione di una entità e delle sue caratteristiche è contenuta in un file XML con estensione *.csdl* e viene chiamata *Conceptual Model*, mentre il modo in cui i dati sono memorizzati nella base dati viene chiamato *Storage Model* ed anch'esso viene memorizzato in un file XML questa volta con estensione *.ssdl*.

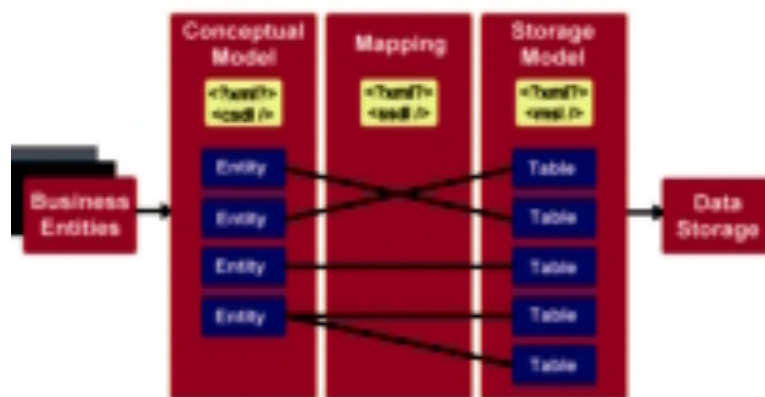



Fig. 2: L'Entity Data Model

Tra questi due modelli, il primo rivolto alla logica di business ed il secondo al data storage, troviamo infine lo schema di *mapping* che definisce il modo in cui le proprietà delle entità sono memorizzate nella base dati. Anche il file di mapping è un file XML che ha estensione *.msl*.

IL CONCEPTUAL MODEL

Un tipico file *csdl* che definisce il *Conceptual Model* è il seguente:

```
<?xml version="1.0" encoding="utf-8"?>
<Schema
 xmlns:cg="http://schemas.microsoft.com/ado/2006/
                               04/codegeneration"
xmlns:edm="http://schemas.microsoft.com/ado/200
                                       6/04/edm"
xmlns="http://schemas.microsoft.com/ado/2006/04/
                                            edm"
       Namespace="NorthwindLib" Alias="Self">
...
</Schema>
```

All'interno di questo schema vanno inserite le definizioni delle entità attraverso il tag **EntityType**, di cui di seguito vediamo un esempio concreto:

```
<EntityType Name="Employee" Key="EmployeeID">
   <Property Name="EmployeeID" Type="Int32"
                              Nullable="false" />
   <Property Name="LoginID" Type="String"
              Nullable="false" MaxLength="256" />
   <Property Name="Title" Type="String"
               Nullable="false" MaxLength="50" />
   <Property Name="BirthDate" Type="DateTime"
                              Nullable="false" />
   <Property Name="HireDate" Type="DateTime"
                               Nullable="false"/>
```

```
<Property Name="SalariedFlag" Type="Boolean"
                                Nullable="false" />
<Property Name="VacationHours" Type="Int16"
                                Nullable="false" />
<Property Name="SickLeaveHours" Type="Int16"
                                Nullable="false" />
<Property Name="CurrentFlag" Type="Boolean"
                                Nullable="false" />

<!-- Altre proprietà -->

<NavigationProperty Name="Contact"
    Relationship="Self
                  .FK_Employee_Contact_ContactID"
    FromRole="Employee" ToRole="Contact" />
<NavigationProperty
                  Name="Managed_Employees"
    Relationship="Self
                .FK_Employee_Employee_ManagerID"
    FromRole="Employee"
                  ToRole="ManagedEmployee" />
</EntityType>
```

In questo esempio viene definita l'entità *Employee* (impiegato) avente *EmployeeID* come chiave univoca specificata con l'attributo *Key*. Questa entità possiede una serie di proprietà che la caratterizzano. Ogni proprietà ha un nome (*Name*), un tipo (*Type*) che viene definito con i tipi dell'*EDM* ed un flag che indica se la proprietà può essere null (**Nullable**) oltre eventualmente ad altri attributi che definiscono meglio la proprietà come ad esempio **MaxLength** per la lunghezza massima in caratteri. I tipi delle proprietà sono forniti dall'*EDM* il quale internamente li traduce nei corrispondenti tipi del CLR e del database utilizzato.
Oltre alle proprietà, nel file *csdl* troviamo anche una serie di relazioni definite attraverso il tag **NavigationProperty**. Nell'esempio in pratica l'entità *Employee* viene messa in relazione prima alla entità *Contact* e poi a se stessa tramite il campo *ManagerID*, attraverso l'utilizzo di una association (il cui nome è specificato dall'attributo **Relationship**) ovvero un'associazione che definisce il rapporto *parent-child* delle due entità coinvolte nella relazione e la modalità con cui questa relazione deve essere gestita. Subito dopo la definizione dell'entità troveremo quindi le due associazioni richiamate dalle due **NavigationProperty**:

```
<Association
        Name="FK_Employee_Contact_ContactID">
    <End Role="Contact" Type="Self .Contact"
                        Multiplicity="1..1" />
    <End Role="Employee" Type="Self.Employee"
                        Multiplicity="1..*" />
</Association>
```

e

```
<Association
        Name="FK_Employee_Employee_ManagerID">
    <End Role="Employee" Type="Self.Employee"
                        Multiplicity="0..1" />
    <End Role="ManagerEmployee"
        Type="Self.Employee" Multiplicity="*" />
</Association>
```

Nel primo caso relativo alla relazione tra *Employee* e *Contact*, l'associazione definisce una relazione di tipo uno a molti per l'entità *Employee* attraverso l'attributo *Multiplicity* impostato ad "1..*" che significa detto a parole: "un *Employee* può avere più *Contact*" (l'asterisco ha il significato di "molti"). Mentre al contrario un *Contact* può essere relazionato ad un solo *Employee*, quindi il suo attributo **Multiplicity** è impostato ad "1..1". In questo modo il **tag Association** ha definito la relazione parent-child esistente tra le due entità *Employee* e *Contact*. Nel secondo caso, la relazione non coinvolge due entità distinte ma è rivolta su se stessa in quanto il manager (ovvero il responsabile) di ogni *Employee* è a sua volta un *Employee*. In questo particolare caso quindi l'association definisce un rapporto molti-a-molti dove ogni *Employee* può avere uno o più *Manager* e viceversa.

LO STORAGE MODEL

I file ssdl contenenti lo *Storage Model* sostanzialmente descrivono la base dati utilizzata dall'applicazione che fa uso dell'*Entity Data Model*. Anche in questo file troviamo le definizioni delle entità e delle associazioni delle stesse tra di loro ed il modo in cui vengono definite è molto simile a quello dei file *csdl*, la differenza sta nei tipi utilizzati che mentre nei file *csdl* sono quelli dell'*Entity Data Model*, nei file *ssdl* sono quelli specifici del database utilizzato dall'applicazione.
La prima parte di un file *ssdl* contiene, come per i file *csdl*, la definizione del *namespace*:

```
<?xml version="1.0" encoding="utf-8"?>
<Schema Namespace="NorthwindModel" Alias="Self"
xmlns="http://schemas.microsoft.com/ado/2006/04/
                                     edm/ssdl">
```

Successivamente troviamo quindi le defini-

ADO.NET ENTITY FRAMEWORK TOOLS

Insieme all'Entity Framework se si utilizza Visual Studio 2008 Beta 2 può essere interessante anche installare il pacchetto ADO.Net Entity Framework Tools Aug 07 Community Technology Preview per aggiungere al designer di Visual Studio il supporto dell'Entity Framework e poter così lavorare in maniera visuale sul codice.

zioni delle entità e delle associazioni:

```
<EntityType Name="Employee" Key="EmployeeID">
    <Property Name="EmployeeID" Type="int"
                                        Nullable="false" />
    <Property Name="LoginID" Type="varchar"
                         Nullable="false" MaxLength="256" />
    <Property Name="Title" Type="varchar"
                          Nullable="false" MaxLength="50" />
    ....
</EntityType>

 <EntityContainer Name="dbo">
    <EntitySet Name="Employee"
EntityType="Self.Employee" Schema="dbo"
                                    Tables="Employee" />
    <AssociationSet
            Name="FK_Employee_Contact_ContactID"
Association="Self.FK_Employee_Contact_ContactID">
            <End Role="Contact" EntitySet="Employee"
                                                      />
            <End Role="Employee"
                                EntitySet="Employee" />
    </AssociationSet>
    <AssociationSet
          Name="FK_Employee_Employee_ManagerID"
Association="Self.FK_Employee_Employee_ManagerI
                                                   D">
            <End Role="Employee"
                                EntitySet="Employee" />
            <End Role="ManagerEmployee"
                                EntitySet="Employee" />
    </AssociationSet>
</EntityContainer>
```

Come possiamo notare i due schemi sono molto simili ma balzano subito agli occhi i diversi tipi utilizzati per le proprietà dell'entità. Infatti se nel file *csdl* avevamo i tipi *Int32, String*, ecc, nel file *ssdl* troviamo invece *int, varchar*, ecc, che sono specifici della base dati considerata, in questo caso un database di SQL Server.

IL MAPPING

Infine abbiamo i file di *mapping* con estensione *.msl* che mettono in relazione il *Conceptual Model* con lo *Storage Model* e si presentano come segue:

```
<?xml version="1.0" encoding="utf-8"?>
<Mapping Space="C-S" xmlns="urn:schemas-
        microsoft-com:windows:storage:mapping:CS">
            <EntityContainerMapping
                        CdmEntityContainer="Employee"
StorageEntityContainer="NorthwindEmployee">
```

```
                    ....
        </EntityContainerMapping>
        ....
</Mapping>
```

Con il tag **EntityContainerMapping** e i due attributi **CdmEntityContainer** e **StorageEntityContainer** mettiamo quindi in relazione la definizione concettuale dell'entità *Employee* (definita con il file csdl) con la definizione fisica della stessa *Employee* nel database **Northwind** (definita con il file *ssdl*). Nel file di mapping, all'interno del tag **EntityContainerMapping** possiamo trovare poi una serie di tag **EntitySetMapping** che mettono in relazione con le tabelle del database eventuali *EntitySet*, ovvero dei raggruppamenti di entità mappate sulla stessa tabella.Fortunatamente con *Visual Studio 2008 Beta 2* anche nelle versioni *Express* gratuite, non è necessario conoscere le specifiche di questi tre file XML di definizione in quanto insieme all'*Entity Framework* viene rilasciato anche un tool denominato *EdmGen.exe (Entity Data Model Generator)* che crea automaticamente per noi questi tre file a partire da un database esistente. Volendo quindi utilizzare per la nostra applicazione il famoso database *Northwind* come abbiamo fatto sino ad ora, sarà sufficiente eseguire dal prompt dei comandi:

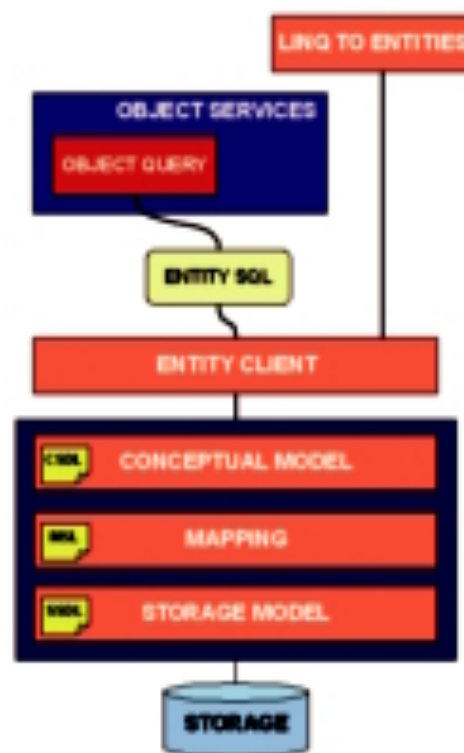



Fig. 3: Panoramica riepilogativa dell'Entity Framework

NOTA

ENTITYSQL

Il nuovo linguaggio SQL dedicato all'Entity Framework e denominato EntitySQL è stato progettato specificatamente per trattare oggetti ed entità a differenza del linguaggio SQL classico che invece permette di accedere e manipolare dati relazionali provenienti da database. Per una trattazione più approfondita di questo nuovo linguaggio è possibile consultare il seguente post del blog ufficiale di ADO.NET:
http://blogs.msdn.com/adonet/archive/2007/05/30/entitysql.aspx

```
edmgen.exe /provider:System.Data.SqlClient

/connectingString="Server=.;database=northwind;int
egrated security=sspi"
/mode:FullGeneration
           /project:Northwind
```

Al termine dell'elaborazione, *EdmGen.exe* avrà generato i tre file menzionati sopra, ovvero i file *Northwind.ssdl*, *Northwind.csdl* e *Northwind.msl* che potranno essere inclusi nel progetto dell'applicazione ed utilizzati tramite l'*Entity Framework*. Oppure dopo aver installato gli *ADO.Net Entity Framework Tools* ancora più semplicemente potremo utilizzare per questa operazione il designer di Visual Studio e l'item template per la creazione di un nuovo *Entity Data Model* con il relativo wizard. Ecco uno schema riepilogativo delle relazioni che intercorrono tra i diversi componenti dell'*Entity Framework*:

CONNECTION STRING

Affinché l'*Entity Framework* possa dialogare con il database ed utilizzarlo quale contenitore per i dati dell'applicazione è necessario creare una *Entity Connection*, ovvero una connection string particolare che utilizza come data provider la dll *System.Data.EntityClient*:

```
<connectionStrings>
       <add name="Northwind"
               connectionString='Metadata=.;

               Provider=System.Data.SqlClient;
               Provider Connection
                               String="server=servername;

               database=Northwind;Integrated
                                          Security=true;

               Connection
            Timeout=10;multipleactiveresultsets=true"'
providerName="System.Data.EntityClient"/>
 </connectionStrings>
```

Notiamo in particolare l'attributo *Metadata* attraverso il quale viene specificata la cartella nella quale sono presenti i tre file dei metadati che abbiamo descritto precedentemente, ovvero il file *.csdl*, *.ssdl* e *.msl*. Con il "." diciamo che i file dei metadati sono presenti nella stessa cartella dell'eseguibile. Se così non fosse dovremmo specificare il percorso completo in cui sono stati salvati i file dei metadati.

ESEGUIRE QUERY SULLE ENTITÀ

Vediamo ora, dopo aver definito i metadati, come possiamo accedere da codice alle entità per interrogarle o modificarle. Innanzi tutto dobbiamo inserire alcune *using* nella nostra classe per accedere più agevolmente ai diversi oggetti di cui faremo uso. Quindi aggiungiamo oltre ai classici namespace anche i seguenti:

```
using System.Data;
using System.Data.Metadata.Edm;
using System.Data.Common;
using System.Data.Objects;
using System.Data.EntityClient;
using System.Linq;
using System.Xml.Linq;
using NorthwindLib;
```

e quindi la nostra classe:

```
public abstract partial class Employee
{
    public ObjectQuery<Customer> AllCustomers
    {
        get
        {
            NorthwindEntities context = new
                                NorthwindEntities();

            var query = from c in context.Customers
                        where c.Orders.Any(o =>
                   o.EmployeeID == this.EmployeeID)
                        select c;

            return query as ObjectQuery<Customer>;
        }
    }
}
```

Alla classe *Employee* già esistente perché definita dall'*Entity Data Model*, aggiungiamo una proprietà che ci restituisce tutti i clienti del dipendente il cui *EmployeeID* corrisponde a quello specificato nella omonima proprietà *EmployeeID* della classe stessa. Quello che viene fatto è innanzi tutto creare una classe parziale in modo da poter aggiungere una nuova proprietà e poi viene creato un oggetto di contesto (*context*) utilizzando la classe *NorthwindEntities* che espone tutte le entità definite nel nostro modello concettuale. Nello specifico di questo esempio, creiamo una query utilizzando *LINQ to Entities*, dove diciamo di recuperare tutte le entità Customer aventi degli ordini la cui proprietà *EmployeeID* corrisponde all'*EmployeeID* cor-

REFERENZIARE DLL NEI PROGETTI

In Visual Studio per referenziare un assembly in un progetto è sufficiente cliccare con il tasto destro del mouse sul nome del progetto, scegliere la voce Add Reference... , cliccare sul tab Browse e selezionare dal disco rigido la DLL che s'intende referenziare nel progetto.

ADO.NET SAMPLES

All'indirizzo: http://www.codeplex.com/adonetsamples/ **è disponibile un nutrito pacchetto di esempi incentrati sull'Entity Framework e con i quali è possibile impratichirsi ulteriormente su questa tecnologia.**

rente. Quindi restituiamo il risultato della query sotto forma di oggetto *ObjectQuery* che contiene l'elenco delle entità *Customer* richieste. Alla query *LINQ* è possibile sostituire il nuovo linguaggio *eSQL (Entity SQL)* sviluppato appositamente per le interrogazioni di dati all'interno dell'*Entity Data Model*. Ecco come si presenterebbe la proprietà *AllCustomers* con l'utilizzo di *eSQL*:

```
public ObjectQuery<Customer> AllCustomers
{
   get
   {
      NorthwindEntities context = new
                                    NorthwindEntities();

      ObjectQuery<Customer> query =
                        context.CreateQuery<Customer>(
         "SELECT VALUE c FROM Orders AS c WHERE
                        c.EmployeeID = @EmployeeID",
         new ObjectParameter("EmployeeID",
                                    this.EmployeeID));

      return query as ObjectQuery<Customer>;
   }
}
```

Infine per recuperare gli oggetti restituiti dalla query è sufficiente ciclare all'interno dell'oggetto *ObjectQuery*:

```
foreach (Customer c in query)
{
   Response.Write(c.CompanyName);
}
```

Aggiungere una nuova entità nella base dati è altrettanto semplice:

```
Category newCat = new Category();
newCat.CategoryName = "Nuova categoria";
context.AddToCategories(newCat);
context.SaveChanges();
int newCatID = newCat.CategoryID;
```

In questo caso aggiungiamo una nuova categoria nella tabella delle categorie del database *Northwind*. Come vediamo è sufficiente creare un oggetto di tipo *Category*, valorizzarlo con i dati che ci interessano e quindi richiamare il metodo **AddToCategories** del contesto

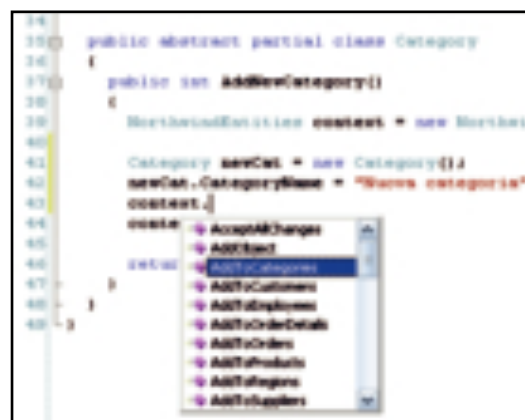



Fig. 4: Metodi dell'oggetto di contesto

per passare il nuovo oggetto appena creato. Subito dopo richiamiamo il metodo **SaveChanges** per persistere effettivamente le informazioni, quindi possiamo leggere la proprietà **CategoryID** della nuova entità aggiunta per leggerne l'ID. Come vediamo dall'immagine seguente, oltre agli specifici metodi per l'aggiunta di ciascuna entità, vi è anche il metodo generico **AddObject** che permette di aggiungere una qualsiasi entità tra quelle previste dal *Conceptual Model*.

CONCLUSIONI

Gestire correttamente i dati nelle applicazioni software è importante in quanto i dati rappresentano il fulcro attorno a cui ruota tutta la logica applicativa. Saper utilizzare quindi strumenti che facilitano la gestione degli stessi e rendono anche più rapido lo sviluppo è importante per produrre software di qualità. E' evidente che al termine della fase Beta dell'Entity Framework gli strumenti messi a disposizione saranno più maturi e permetteranno allora realmente di creare delle solide basi per lo sviluppo di tutte quelle applicazioni che si basano fortemente sui dati e sulla loro manipolazione. Non ci resta quindi che attendere il rilascio della versione definitiva de—ll'Entity Framework per cominciare ad utilizzarlo al pieno delle sue potenzialità.

Gianni Malanga

L'AUTORE

Gianni Malanga Lavora da più di 10 anni con tecnologie Microsoft in particolare nel campo dello sviluppo Web. Sviluppa in .NET ormai da alcuni anni e ha svolto diverse docenze per corsi di formazione professionali. Dopo aver lavorato alle dipendenze di importanti realtà locali, attualmente ha intrapreso la libera professione costituendo la ditta individuale Kaone Consulting che si occupa di consulenza informatica per PMI su tutto il territorio nazionale. Gli si può scrivere all'indirizzo

ADO.NET ENTITY FRAMEWORK BETA 2

L'Entity Framework Beta 2 è disponibile doppia versione per processori a 32 bit (EFSetup-x86.exe) e a 64 bit (EFSetup-x64.exe) al seguente indirizzo del sito Microsoft:
http://www.microsoft.com/downloads/details.aspx?FamilyID=f1adc5d1-a42e-40a6-a68c a42ee11186f7&displaylang=en

Tra i requisiti dell'Entity Framework vi è la presenza sulla macchina del .NET Framework 3.5 Beta 2 che è possibile recuperare da questo indirizzo:
http://www.microsoft.com/downloads/details.aspx?FamilyID=d2f74873-c796-4e60-91c8-f0ef809b09ee&DisplayLang=en

I trucchi del mestiere

Tips & Tricks

Questa rubrica raccoglie trucchi e piccoli pezzi di codice, frutto dell'esperienza di chi programma, che solitamente non trovano posto nei manuali. Alcuni di essi sono proposti dalla redazione, altri provengono da una ricerca su Internet, altri ancora ci giungono dai lettori. Chi volesse contribuire, potrà inviare i suoi Tips&Tricks preferiti. Una volta selezionati, saranno pubblicati nella rubrica. Il codice completo dei tips è presente nel CD allegato nella directory *tips*\\ o sul Web all'indirizzo: cdrom.ioprogrammo.it.

COME POSSO IMPLEMENTARE IL BUBBLE SORT

```
#include <iostream>
using namespace std;
int compare(int, int);
void sort(int[], const int);
void swap(int *, int *);
int compare(int x, int y)
{
    return(x > y);
}
void swap(int *x, int *y)
{
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
}

void sort(int table[], const int n)
{
    for(int i = 0; i < n; i++)
    {
        for(int j = 0; j < n-1; j++)
        {
            if(compare(table[j], table[j+1]))
                swap(&table[j], &table[j+1]);
        }
    }
}

int tab[3];

int main()
{
cout << "Input numbers: \n";
for (int i = 0; i < 3; i++)
{
    cin >> tab[i];
}

cout << "Before sorting: ";
for (int i = 0; i < 3; i++)
{
    cout << tab[i] << " ";
}

cout << "\nAfter sorting: ";
sort(tab, 3);
for(int i = 0; i < 3; i++)
{
    cout << tab[i] << " ";
}
return 0;
}
```

COME POSSO CONVERTIRE UN NUMERO DA ARABO IN ROMANO

```
public String deka2r( int nr )
  {
    String deka2r = "";
    try
    {
      if( nr >= 889 )
      {
        deka2r = "M" + deka2r( (nr - 1000) );
      }
      else if( nr >= 389 )
      {
        deka2r = "D" + deka2r( (nr - 500) );
      }
      else if( nr >= 89 )
      {
        deka2r = "C" + deka2r( (nr - 100) );
      }
```

```
      else if( nr >= 39 )
      {
        deka2r = "L" + deka2r( (nr - 50) );
      }
      else if( nr >= 9 )
      {
        deka2r = "X" + deka2r( (nr - 10) );
      }
      else if( nr >= 4 )
      {
        deka2r = "V" + deka2r( (nr - 5) );
      }
      else if( nr >= 1 )
      {
        deka2r = "I" + deka2r( (nr - 1) );
      }
      else if( nr <= -889 )
      {
        deka2r = "M" + deka2r( nr + 1000 );}
      else if( nr <= -389 )
      { deka2r = "D" + deka2r( nr + 500 )      }
      else if( nr <= -89 )
      {
        deka2r = "C" + deka2r( nr + 100 );
      }
      else if( nr <= -39 )
      {
        deka2r = "L" + deka2r( nr + 50 );
      }
      else if( nr <= -9 )
      {
        deka2r = "X" + deka2r( nr + 10 );
      }
      else if( nr <= -4 )
      {
        deka2r = "V" + deka2r( nr + 5 );
      }
      else if( nr <= -1 )
      {
        deka2r = "I" + deka2r( nr + 1 );
      }
    }
    catch(Exception _e_) { Err.set(_e_,"deka2r"); }
    return deka2r;
  }
```

VB.NET

COME POSSO EVITARE DI APRIRE DUE ISTANZE DELLA STESSA APPLICAZIONE?

```
'Windows API
Declare Function OpenIcon Lib "user32" (ByVal hwnd As Long) As Long
Declare Function SetForegroundWindow Lib "user32" (ByVal hwnd As
                                                        Long) As Long

''' <summary>
''' Function to check and see if an instance of the application is already
                                                                    open
''' </summary>
''' <param name="sApp">Name of the application</param>
''' <returns>True/False</returns>
''' <remarks></remarks>

Public Shared Function IsAlreadyOpen(ByVal sApp As String) AsBoolean
   'Check running processes to see if application is already running

   Dim pProcess As Process() =
                  System.Diagnostics.Process.GetProcessesByName(sApp)
   If pProcess.Length > 1 Then  'If > 1 then its already running
      Return True
   Else  'Not running
      Return False
   End If
End Function

''' <summary>
''' Procedure to activate current instance
''' if user is trying to open a 2nd instance of application
''' </summary>
''' <remarks></remarks>

Public Shared Sub ActivatePrevInstance()
   Dim lngPrevHndl As Long
   Dim lngResult As Long

   Dim oProcess As New Process 'Variable to hold individual Process
   lngPrevHndl = oProcess.MainWindowHandle.ToInt32()
   If lngPrevHndl = 0 Then
      Exit Sub 'if No previous instance found exit the the procedure
   Else
      ''If found
      lngResult = OpenIcon(lngPrevHndl) 'Restore the program.
      lngResult = SetForegroundWindow(lngPrevHndl) 'Activate the
                                                        application.

      End 'End the current instance of the application.
   End If
End Sub
'Sample usage:

'Call from main form of you application in the Form_Load event

If Class1.IsAlreadyOpen(Application.ProductName) Then

    MessageBox.Show("You cannot have 2 instances of " & _
    Application.ProductName & " running, closing application", & _
    "Error: Application already open", & _
    MessageBoxButtons.OK, MessageBoxIcon.Error)
    Class1.ActivatePrevInstance()

End If
```

VALIDARE L'INPUT USANDO FLEX

AFFRONTIAMO UNO DEI PROBLEMI BASILARI PER OGNI PROGRAMMATORE: CONTROLLARE LA CORRETTA IMMISSIONE DEI DATI. COME ESEMPIO APPLICATIVO UTILIZZEREMO UNA FORM FLASH PER L'AUTENTICAZIONE DEGLI UTENTI

Nella puntata precedente ci siamo lasciati con una applicazione che simulava la procedura di login al "Music Shop". Infatti qualunque cosa inserissimo come user e password , l'applicazione rispondeva con una semplice finestra di messaggio. Vediamo allora come possiamo migliorare la nostra applicazione, sia da un punto di vista funzionale che architetturale.

VALIDAZIONE DELL'INPUT

Partiamo dal punto di vista funzionale. Una prima cosa che banalmente l'applicazione, o meglio, nello specifico, la maschera di login deve fare è controllare che i campi relativi alla users e/o alla password non siano vuoti quando si preme il pulsante "Conferma". Come si può facilmente immaginare, non è conveniente far effettuare questi controlli al codice lato server, in quanto sprecheremmo un round-trip per verificare che magari il campo è vuoto. Come possiamo controllare queste informazioni prima dell'invio ? Una soluzione banale potrebbe essere quella di effettuare le verifiche nel metodo associato al click del pulsante. Questa soluzione decisamente funziona, ma possiamo fare di meglio utilizzando una feature del Flex SDK, cioè i *validators*.
Il Flex SDK mette a disposizione una serie di classi di validazione, dette appunto *validators*, per effettuare la convalida di informazioni inserite dall'utente. Si va dalla validazione di una informazione obbligatoria, alla validazione di un indirizzo e-mail piuttosto che di un valore numerico o di una data. Alcuni validators possono risultare non appropriati se applicati in un contesto prettamente nazionale, ad esempio l' SSN validator è rivolto a validare il codice sanitario americano. Comunque, indipendentemente da questo, vi è sempre la possibilità di derivare una classe customizzata ed implementare un meccanismo di validazione che si adatta maggiormente alle esigenze della applicazione che si sta sviluppando. Iniziamo con il vedere come validare semplicemente le due informazioni immesse nei campi relativi allo user ed alla password. Per definire un validator utilizzando la sintassi mxml utilizziamo il tag:

```
<mx:Validator id="reqValid" required="true"
        source="{txtUserName}"
        property="text"
invalid="m_ml.handleValidation(event)" />
```

Oltre ad indicare l'id dell'elemento mxml, và indicato il controllo da validare e la sua specifica proprietà da verificare. Nel nostro caso stiamo validando lo username, cioè stiamo validando ciò che si trova nel campo text di un controllo *TextInput*. Quindi utilizzando le proprietà del controllo *Validator, source e property*, indichiamo nello specifico cosa sottoporre a validazione.
Nell'esempio abbiamo anche utilizzato l'evento *invalid*. Questa indica una azione da intraprendere in seguito al fallimento della validazione. Per contro esiste anche un evento valid a cui associare operazioni necessarie in caso di una validazione riuscita. Normalmente Flex esegue la validazione quando viene sollevato l'evento *valueCommit* di un controllo. Ciò avviene ad esempio quando il controllo perde il focus, cioè l'utente si sposta dal controllo in questione su di un altro, oppure se la proprietà osservata viene modificata da codice. E' possibile modificare questo comportamento standard utilizzando la proprietà *trigger* e l'evento *triggerEvent*. Così facendo possiamo indicare che la validazione deve avere luogo quando l'evento indicato da *triggerEvent* del controllo indicato da *trigger* viene sollevato.
Ad esempio:

```
<mx:Validator id="reqValid" required="true"
        source="{txtUserName}"
        property="text"
invalid="m_ml.handleValidation(event)"
```

```
trigger="{btnConfirm}"
triggerEvent="click"/>
```

In questo modo abbiamo indicato che la validazione deve avere luogo in seguito all'evento *click* del controllo con id *btnConfirm*, cioè il nostro pulsante "Conferma".
Riassumendo, il seguente codice:

```
<mx:ControlBar
horizontalAlign="center">
<mx:Button
label="Conferma" id="btnConfirm" click=""/>
<mx:Button
label="Annulla" id="btnCancel" cornerRadius="5"/>
</mx:ControlBar>

<mx:Validator id="reqValid" required="true"
source="{txtUserName}" property="text"
invalid="m_ml.handleValidation(event)"
trigger="{btnConfirm}" triggerEvent="click"
valid="SendData()" />
```

permette di eseguire la validazione sulla pressione del pulsante "Conferma" e di ottenere una segnalazione di errore per l'utente come indicato in Figura 1.

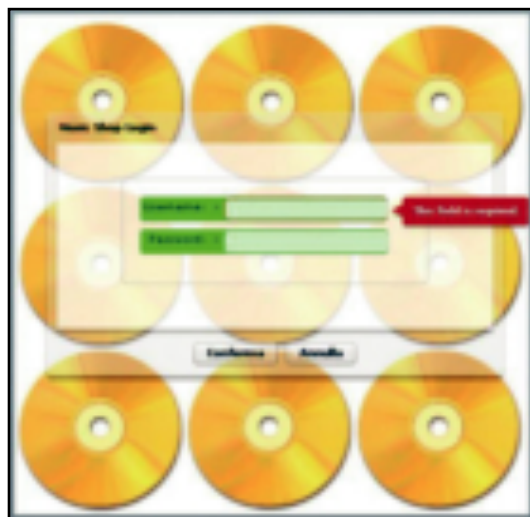



Figura 1: Segnalazione dell'errore da parte del validator.

È chiaramente possibile modificare il testo del messaggio, altrimenti un pò laconico, in modo da indicare, all'utente, il problema con maggiore precisione. Per far ciò, basta utilizzare la proprietà *requiredFieldError*. Per poter effettuare una convalida più *fine*, nel caso di una stringa di testo, possiamo usare il tag *<mx:StringValidator>*. Con questo tipo di validator, potremo ad esempio, indicare una lunghezza minima e massima per la stringa e i relativi messaggi da visualizzare all'utente in caso di fallimento della verifica.

PERSONALIZZARE LA VALIDAZIONE

Nel caso in cui nessuno dei validators messi a nostra disposizione dal Flex SDK sia adatto alla situazione specifica, possiamo scriverne uno custom, in maniera molto semplice. Consideriamo il caso in cui dobbiamo verificare che la password, ad esempio, rispetti una serie di precise regole:

- Non contenga lo username;
- Contenga almeno un valore maiuscolo;
- Contenha almeno un valore numerico;

In questo caso, non avendo un validator preconfezionato possiamo svilupparlo noi semplicemente derivandolo dalla classe Validator, vedi figura 2.

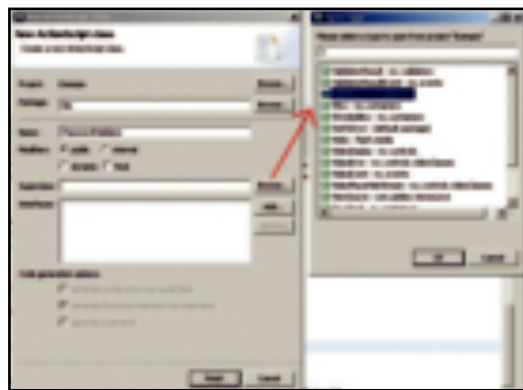
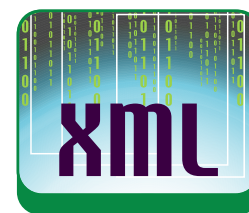



Figura 2: Creaimo la nostra classe di validazione derivandola da mx.validators.

Il codice completo del validator custom lo trovate nel progetto allegato, nel file PasswordValidator, nella cartella "Biz\Validators". La cosa da sottolineare è che tutto quello che dobbiamo fare per definire il nostro validator è effettuare solo l'override del metodo *doValidation*. Infatti esso sarà poi richiamato dal framework durante il processo di validazione.
Il validator cosi realizzato potrà essere utilizzato sia come tag mxml che richiamato via codice. Per utilizzarlo come tag mxml, è necessario specificare nel tag *mx:Application* il namespace nel quale è dichiarata la classe del validator stesso.
Quindi inserendo nel tag Application:

```
xmlns:BizVldts="Biz.Validators.*"
```

indichiamo con la keyword *BizVldts* un alias per il namespace *Biz.Validators* e possiamo cosi riferirci al validator custom come se fosse un normale tag mxml:

```
<BizVldts:PasswordValidator id="pwdValid"
source="{txtPassword}"
property="text"
```

NOTA

COME INIZIARE

Per cominciare subito a lavorare, copiate il codice allegato all'articolo sul vostro HD. Poi aprite il Flex Builder, quindi dal menu File | New, scegliete Import e poi la voce Existing Projects into Workspace. Indicate la cartella che contiene il file .mxml principale del progetto, ed il gioco è fatto.

```
required="true"
UserName="{txtUserName.text}"/>
```

VALIDAZIONE MULTIPLA

Dato che il nostro processo di validazione si compone in effetti di due validazioni, la prima per lo username e la seconda per la password, e se concluso con successo, deve inviare queste informazioni al server per l'autenticazione, possiamo riorganizzare il nostro codice in questo modo:

```
<mx:Button label="Conferma" id="btnConfirm"
click="ValidateForm()" />
```

Il pulsante conferma richiama un metodo *ValidateForm* in cui eseguire i metodi *validate* dei singoli validators:

```
private function
ValidateForm():void
{
result =
userValid.validate();
if(result.type ==
ValidationResultEvent.INVALID) return;

result =
pwdValid.validate();
if(result.type ==
ValidationResultEvent.INVALID) return;

SendData();
}
```

Cosi, solo quando entrambi i validators sono eseguiti con successo, verrà richiamato il metodo SendData per l'invio dei dati al server. Questo è necessario in quanto se, come visto prima, indichiamo per ciascun validator le informazioni di trigger ed impostiamo la proprietà *valid*, alla prima validazione positiva il metodo specificato nella proprietà *valid* sarà eseguito, mentre nel frattempo si procede con la seconda validazione. Questo può rendere ambigua l'operazione di validazione e generare uno spreco di risorse in quanto sarà avviata la comunicazione con il server durante una fase intermedia della validazione.

NOTA

SCARICARE FLEXBUILDER

Il Flex Builder ed il Flex SDK possono essere scaricati dal sito della Adobe all'indirizzo: http://www.adobe.com/products/flex/. Chiaramente, l'installazione del Flex Builder include quella del Flex SDK.

CONFIGURAZIONE DELL'APPLICAZIONE

Generalmente una applicazione ha bisogno di una serie di informazioni con cui configurarsi sin dal primo avvio. Per realizzare un file di configurazione per il nostro Music Shop possiamo pensare ad un piccolo file .xml contenente le informazioni necessarie allo start-up dell'applicazione. Ad esempio, l'immagine di sfondo e gli url per l'autenticazione, in modo da non dover scrivere queste informazioni nel codice e poterle modificare con un semplice editor di testo.
Il nostro file di configurazione sarà molto semplice, ma ci permetterà comunque di introdurci al supporto dedicato dal Flex SDK alla gestione dell'XML. Nel file di configurazione troviamo solo le seguenti informazioni:

```
<MusicShop>
<LoginUrl url="
http://localhost/MusicShop/Login.aspx" />
<BackgroundImage>
F:\Progetti\Articoli\Flex2\Immagini\CDWall.jpg
</BackgroundImage>
</MusicShop>
```

Strutturato in questa maniera, il file .xml presenta una informazione, l'url della pagina server per l'autenticazione, sotto forma di attributo, mentre il nome dell'immagine di sfondo è costituita da un nodo. Vediamo come poterlo analizzare.
Nell'ottica di organizzare al meglio il codice, prepariamo una nuova classe chiamata "Config" che deriva dalla classe *EventDispatcher*. La classe base EventDispatcher consente all'oggetto derivato di accedere a tutta una serie di funzionalità legate al modello degli eventi del Flex Framework. Tralasciando i particolari della gestione degli eventi su cui si soffermeremo in seguito, quello che ci interessa al momento della classe Config è la sua capacità di leggere il file .xml di configurazione ed accedere alle sue parti.
Nel metodo *Load* abbiamo definito il meccanismo di lettura del file. Infatti mediante i due oggetti *URLRequest* ed *URLLoader* possiamo caricare una sorgente dati esterna, nello specifico un file .xml. La classe URLRequest permette di catturare le informazioni presenti in uno stream HTTP definito verso l'url che specifichiamo nel costruttore della classe. Quindi combinando questa con la classe URLLoader, che si occupa dell'accesso all'url prima specificato, è possibile effettuare il caricamento di dati da una qualsiasi sorgente esterna, nel nostro caso un file .xml.

```
private var
myLoader:URLLoader;
:
:
var XML_URL:String =
"MusicShop.xml?nocache=" +
```

```
new
Date().getTime();;
var myXMLURL:URLRequest =
new URLRequest(XML_URL);
myLoader = new URLLoader();
```

Per accedere fisicamente ai dati caricati, va definito un event-handler per l'evento *Event.COMPLETE*, che sarà richiamato quando l'operazione di caricamento è terminata. Con la seguente riga:

```
myLoader.addEventListener(Event.COMPLETE,
xmlLoaded);
```

indichiamo che l'event-handler è rappresentato dal metodo xmlLoaded che effettua un casting delle informazioni lette in un oggetto di tipo XML:

```
myXML = XML(myLoader.data);
```

FLEX E L'XML

Con l'introduzione di ActionScript 3.0, l'approccio all'utilizzo di dati in formato xml è stato notevolmente facilitato. In particolare sono state introdotte un insieme di classi e funzionalità che prendono il nome di *ECMAScript for XML*, o più semplicemente *E4X*.
Tra le classi specificate in E4X, quella su cui focalizzeremo l'attenzione e che useremo per leggere le informazioni presenti nel file di configurazione è *XML*.
Come tutte le classi presenti nel paradigma E4X, *XML* è stata progettata per rendere l'accesso ai dati in formato xml intuitiva tramite l'usuale sintassi basata sul (.).
Se diamo un'occhiata al contenuto del file di configurazione, possiamo notare che l'informazione circa l'url per l'autenticazione dei dati è posto in un attributo del nodo LoginUrl, mentre il path all'immagine di sfondo si trova nel nodo *BackgroundImage*.
Tuttavia, benchè effettivamente si tratta di accedere a due elementi xml differenti, vederemo come la sintassi E4X permetta di accedervi in maniera molto simile.
Le seguenti due righe permettono la lettura dei dati:

```
m_url = myXML.LoginUrl.@url;
m_backgroundImage =
myXML.BackgroundImage;
```

Con la prima istruzione, accediamo al nodo LoginUrl, di cui leggiamo il contenuto dell'attributo "url", mediante la sintassi (.) ed l'operatore (@) il quale specifica che, il nome che lo segue immediatamente, è il nome di un attributo.
Accedere invece al contenuto del nodo *BackgroundImage* avviene banalmente con la seconda istruzione.
Si nota, quindi, come sia effettivamente molto semplice accedere agli elementi di una struttura xml, e questa semplicità è mantenuta anche in caso di scrittura.
Per quanto riguarda le nostre informazioni di configurazione noi siamo interessati solo alla loro lettura, ma se ci trovassimo nella condizione di dover persistere la modifica di una di queste, non dovremmo effettuare alcun sforzo.
Infatti se volessimo salvare nell'attributo "url" la stringa "www.google.it", ci basterebbe scrivere:

```
myXML.LoginUrl.@url = "www.google.it";
```

E' possibile anche effettuare lettura o scrittura basate su di un filtro. Supponiamo che il nostro file xml abbia una struttura del tipo:

```
<orders>
  <order id='1'>
    <article>article1</article>
    <price>3.95</price>
  </order>
  <order id='2'>
    <article>article2</article>
    <price>1.45</price>
  </order>
</orders>
```

In lettura possiamo usare la sintassi:

```
myXML.item.(@id==2).article;
myXML.item.(article=="article1").price
```

mentre in scrittura, sugli stessi campi:

```
myXML.item.(@id==2).article = "new article2";
myXML.item.(article=="article1").price = 100.00;
```

ARCHITETTURA DELLA APPLICAZIONE

Una volta superata la pagina di login, cosa si presenterà all'utente riconosciuto ? In base al suo account l'utente potrà essere diretto verso la pagina di gestione del Music Shop, se amministratore, oppure verso la pagina di consultazione dell'elenco cd nel caso di un "cliente".
Impropriamente ho usato il termine "pagina", più adatto ad una applicazione web tradizionale. In effetti, comunque, l'utente si troverà di fronte ad una interfaccia diversa da quella iniziale di login. Come preparare queste due nuove "pagine" ? Vedremo ora come realizzare le due interfacce, in due modi, ognuno con i suoi difetti e pregi.

NOTA

L'ELENCO DEI VALIDATORS

Potete trovare un elenco completo dei validators messi a disposizione dal Flex SDK al seguente indirizzo: http://livedocs.adobe.com/flex/2/langref/mx/validators/package-detail.html

Mentre per un approfondimento sui meccanismi di validazione: http://www.adobe.com/devnet/flex/quickstart/validating_data/

STATI VS MODULI

Per organizzare la nostra interfaccia in modo che si adatti in risposta alla interazione con l'utente, Flex ci mette a disposizione i tags *mx:states* ed *mx:state*. Possiamo aggiungere degli stati alla nostra applicazione in maniera molto semplice. Spostandoci in modalità design, apriamo la vista States dal menu Window | States. Comparirà quindi la finestra *States*, e premendo il pulsante "New State" potremo aggiungere un nuovo stato, per il quale dovremo definire il nome.

La finestra contenente la maschera di login, rappresenta lo stato di base della nostra applicazione. Facendo un riferimento alla programmazione ad oggetti, possiamo immaginare che ciascun stato che andiamo a creare rappresenti una sorta di derivazione dall'interfaccia dello stato base. Infatti, non appena creiamo un nuovo stato, esso a livello di elementi presenti nella maschera, sarà uguale alla maschera di login.

Creiamo quindi uno stato e chiamiamolo "AdminState". Esso comparirà nell'elenco degli stati presente nella finestra degli stati. Se ci spostiamo tra lo stato "AdminState" e quello base, cliccando rispettivamente sul loro nome nell'elenco, notiamo che in design non cambia nulla, tranne per il nome dello stato corrente visualizzato nella parte superiore della design view. Adesso spostiamoci nello stato "AdminState" e cancelliamo tutto ciò che è presente sullo stage. Ripetiamo l'operazione di selezionare alternativamente i due stati ed adesso vediamo che qualcosa è cambiato. Aggiungiamo un altro stato, "UserState", eliminiamo i componenti ereditati, e spostiamoci in source view, per vedere la struttura di tag relativa agli stati.

Questa si presenta così:

```
<mx:states>
    <mx:State name="AdminState">
        <mx:RemoveChild target="{image1}"/>
        <mx:RemoveChild target="{controlbar1}"/>
        <mx:RemoveChild target="{pnlLogin}"/>
    </mx:State>
    <mx:State name="UserState">
        <mx:RemoveChild target="{controlbar1}"/>
        <mx:RemoveChild target="{pnlLogin}"/>
        <mx:RemoveChild target="{image1}"/>
    </mx:State>
</mx:states>
```

La nidificazione degli strati non ha limite, ogni stato può avere una serie di suoi sotto stati e così via. Tra tutti gli stati ve ne uno che è indicato come *Start State*, che rappresenta l'interfaccia che ci viene presentata alla partenza dell'applicazione. Se non definiamo nessuno degli stati creati come *Start State*, il primo stato è per default quello dello stage iniziale o Base State. Lo stato di partenza può essere anche definito mediante la proprietà *currentState* del tag *mx:Application*.

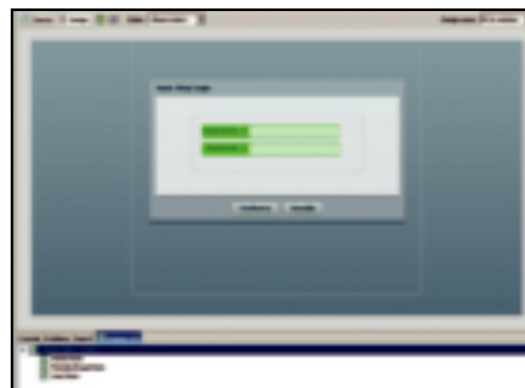



Figura 3: La gerarchia degli stati dopo il nostro intervento.

Creati i due stati, per l'amministratore e l'utente, passiamo a caratterizzarli in qualche modo. Selezioniamo lo stato "AdminState" e posizioniamo su di esso, selezionandolo dal pannello dei Componenti, un Datagrid. Per lo stato "UserState", scegliamo invece un componente DateChooser. Fatto questo scriviamo nel metodo SendData il seguente codice:

```
if((this.txtUserName.text== "Admin") &&
        (this.txtPassword.text == "Admin1"))
{
    this.currentState = "AdminState";
}
else
{
    if((this.txtUserName.text ==
"User") && (this.txtPassword.text == "User1"))
    {
        this.currentState =
                "UserState";
    }
    else
    {
Alert.show("Autenticazione fallita !!");
    }
}
```

Questo, oltre a definire i due utenti al momento riconosciuti dal sistema, permette di cambiare stato, e quindi interfaccia presentata all'utente, in base al riconoscimento effettuato.

Cosa può esserci di svantaggioso nell'utilizzare gli stati ? Come possiamo vedere ritornando in modalità *source view*, tutti gli elementi grafici che inseriamo su ciascuno stato, sono inseriti all'interno del tag *mx:State* relativo. Una interfaccia complessa, con più stati, e magari anche con sottostati, può rendere complesso il codice mxml, se volessimo apportare delle modifiche direttamente in *source view*. Uno scenario

NOTA

XML SUL WEB

Per il supporto di Flex all' XML fare riferimento alla documentazione disponibile all'url:
http://livedocs.adobe.com/flex/2/docs/wwhelp/wwhimpl/common/html/wwhelp.htm?context=LiveDocs_Parts&file=00001910.html

di questo tipo non è poi molto fantasioso, si pensi alla definizione delle informazioni di DataBinding. Per definire le quali potremmo sempre usare le proprietà del controllo, ma magari stiamo definendo variabli e metodi nel codice e quindi ci viene naturale spostarci nella parte relativa ai tag per completare gli agganci. Un altro svantaggio è la effettiva non modularizzazione dell'architettura. In fondo, anche se logicamente, i diversi stati sono definiti come separati, essi si trovano sempre codificati nello stesso file fisico.

I MODULI

Una alternativa agli stati, nella implementazione della architettura di sviluppo, è rappresentata dai cosidetti moduli. Questi non sono altro che dei file .swf che possono essere caricati, da una o più applicazioni, e "scaricati" quando non più necessari. Qui di seguito vedremo come realizzare ed utilizzare i moduli, ma per una trattazione più approfondita dell'argomento vi rimando al link indicato nel box laterale. Il Flex Builder non mette a disposizione un meccanismo semplice per la loro creazione come avviene nel caso di una applicazione MXML. Però, dato che un modulo, in termini di codice mxml, è molto simile ad una applicazione, possiamo creare una *mx:Application* e poi convertirla in *mx:Module*. Infatti il corpo di un modulo, ridotto ai minimi termini corrisponde al seguente codice:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Module
        xmlns:mx=http://www.adobe.com/2006/mxml
                layout="absolute">

</mx:Module>
```

Una volta creato il modulo, la sua compilazione mediante il Flex Builder, non richiede alcun intervento in quanto il modulo sarà compilato insieme al resto dell'applicazione. Il file .swf prodotto dalla compilazione, non potrà essere fruito se non dall'applicazione principale, quindi richiamarlo in un Flash Player non sortirà effetto.

I moduli, poi, essendo dei file eseguibili, pur con la limitazione appena indicata, devono necessariamente trovarsi nella directory principale, insieme al file dell'applicazione.

Per trasformare la nostra applicazione, da basata sugli Stati a basata sui *Moduli*, aggiungiamo un ulteriore stato chiamato "ModulesStageState", eliminiamo i componenti ereditati e inseriamo al centro dello stage dello stato un componente *mx:VBox*, presente tra i componenti di layout.

Il nuovo stato servirà da contenitore per tutti gli eventuali moduli utilizzati dal Music Shop. Fatto questo, il metodo SendData subirà alcune piccole modifiche in modo che, al riconoscimento dell'utente, sia caricato il modulo corretto. Ad esempio, in caso di utente amministratore:

```
    theModule =
        ModuleManager.getModule("AdminModule.swf");
theModule.addEventListener(ModuleEvent.READY,
                                adminModuleReady);
    theModule.load();
```

Queste tre righe, mediante la classe ModuleManager effettuano il caricamento del modulo "AdminModule.swf". Inoltre essendoci registrati all'evento *ModuleEvent.READY*, quando il modulo è stato caricato, possiamo rendere corrente lo stato "ModulesStageState" e aggiungere alla display list del componente layout VBox l'istanza del modulo, in modo da visualizzarla.

```
    this.currentState = "ModulesStageState";
this.vbAdmin.addChild(theModule.factory.create() as
                                AdminModule);
```

COMUNICAZIONI APPLICAZIONE – MODULI

La comunicazione tra l'applicazione principale ed i suoi moduli è bidirezionale. Infatti dalla applicazione è possibile utilizzare proprietà e metodi di un modulo utilizzando la proprietà *factory* della classe *ModuleManager*. Per accedere ad un metodo del modulo AdminModule, sarà sufficiente accedere all'istanza del modulo e quindi richiamare il metodo:

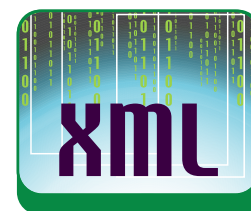
```
        var m:Object =
    theModule.factory.create() as AdminModule;
        m.MetodoPresenteNelModulo();
```

Viceversa, la comunicazione tra modulo ed applicazione avviene quasi banalmente attraverso la proprietà, del modulo, parentApplication:

```
parentApplication.ProprietàDellaApplicazione;
parentApplication.MetodoDellaApplicazione();
```

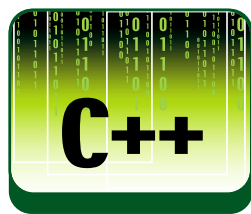
CONCLUSIONI

Due parole in conclusione. Vi lascio ad una lenta digestione dei concetti esposti finora in attesa della prossima puntata in cui si comincerà a lavorare con dati "reali" in quanto, finalmente, stabiliremo una comunicazione con la parte server dell'applicazione Music Shop.

Giuseppe Dattilo

AGGIORNARE FLEX

Per una trattazione esaustiva dell'uso dei Moduli in Flex, scaricate l'aggiornamento del capitolo 31 del manuale del Flex SDK dall'indirizzo http://blogs.adobe.com/flexdoc/pdfs/modular.pdf

DIECI COSE DA NON FARE IN C++

CHI INIZIA COL C++ È SEMPRE PORTATO A FARE GLI STESSI ERRORI. QUESTI A VOLTE PRODUCONO PROGRAMMI MALFUNZIONANTI, BUG CRITICI, O SEMPLICEMENTE LO SCHERNO DA PARTE DEI PIÙ ESPERTI. ECCO COME EVITARLI.

Benvenuti al nostro appuntamento mensile col C++. In questo anno abbiamo analizzato librerie, pratiche e stili da utilizzare per migliorare la programmazione in questo linguaggio potente e complesso. Molto complesso. Soprattutto perché è necessario imparare tanto *cosa fare* in C++ quanto (e soprattutto) cosa ***non*** fare. Puntualmente, svolgendo la mia attività di docente e frequentando qualche gruppo di discussione, mi è capitato di sentire programmatori, soprattutto novizi, ripetere per l'ennesima volta gli stessi errori, o le stesse leggende metropolitane.
In questa puntata ne ho riassunti dieci fra quelli che considero più gravi o più diffusi. Nella maggior parte dei casi si tratta di errori di inesperienza, e il discorso si rivolgerà quindi ai neofiti. Alcuni risvolti, comunque, potrebbero sorprendere più di un programmatore di media esperienza. Se seguite puntualmente questa rubrica, vedrete rispuntare fuori argomenti già trattati in quest'anno: considerate questa puntata come un buon riassunto. Ma potete anche vederla come un buon antipasto: alcune delle indicazioni presenti in questo decalogo, infatti, faranno da spunto per le serie dei prossimi mesi. Ed ora, ecco a voi la top ten degli orrori in C++:

REQUISITI

Conoscenze richieste
Conoscenza di base del C++

Software
Un compilatore C++ standard

Impegno

Tempo di realizzazione

1) "COMPILA, QUINDI È GIUSTO!"

Questo non è un errore, di per sé. Ciò nonostante si conquista senza alcun dubbio la prima posizione in quanto **capostipite di una numerosa famiglia di errori**. Si tratta dell'atteggiamento tipico del neofita, che, un po' per inesperienza e un po' per pigrizia pensa che il fatto che un compilatore accetti una scrittura implichi necessariamente che questa sia corretta. Non è così. L'unica fonte attendibile che stabilisce cosa sia da considerarsi corretto nel C++ è **lo standard**. Non sono attendibili tutte le fonti che non fanno riferimento allo standard e ancor meno il comportamento dei compilatori. I compilatori, infatti, applicano spesso qualche deviazione o estensione rispetto al C++ standard, e **devono** necessariamente adottare qualche arbitrio nell'interpretazione dei famosi **comportamenti dipendenti dall'implementazione, non specificati, o indefiniti**. Facciamo un test pratico, dedicato soprattutto ai lettori esperti:

```
#include <iostream>

int main() {
    signed char n = 127;
    std::cout << int(++n);
}
```

Quale sarà l'output dell'applicazione? Provate a rispondere da voi, prima di continuare la lettura.
Fatto? L'unica risposta corretta è: **"boh?"**. Questo, infatti, è un tipico caso di comportamento indefinito. Non si sa cosa succederà in esecuzione, perché lo standard non stabilisce:

- **Il valore massimo di char**. Il fatto che un char debba occupare al minimo 1 byte e che quasi sempre questa misura sia anche quella effettiva, non significa che un char non possa essere rappresentato con 2, 3 o 10 byte. Potete controllare il valore massimo di un signed char richiamando la funzione std::numeric_limits<signed char>::max(). Se questo valore è maggiore di 127, verrà stampato semplicemente 128. Altrimenti avverrà un overflow.
- **Che cosa succede in caso di overflow di un valore signed**: dato che la stragrande maggioranza delle architetture usa il complemento a due, è probabile che il numero riparta dal valore minimo (-128). Ma anche no. Potrebbe rimanere 127. O l'applicazione potrebbe andare in crash.
Potete inventarvi il comportamento che volete: tanto lo standard non dice nulla a riguardo.

tutti, specialmente negli esempi infidi in un linguaggio insidioso come C++. Il punto è un altro: se lo provate sul vostro compilatore di fiducia e questo stampa "255", o "65535" o arresta l'applicazione, o qualsiasi altra cosa, **non significa che questo sia il comportamento "giusto"**. (In questo caso, infatti, *non* c'è un comportamento "giusto").
Il ricorso allo standard non serve per farvi diventare l'ennesimo *language lawyer* capace di discutere per ore con un collega recitando il paragrafo 57, comma Q, sezione 0xA4 del Libro dei Libri; ha un valore pratico.
Se scrivete software portabile, l'attenzione a non cadere nel comportamento indefinito o a non dare per scontato un comportamento dipendente dall'implementazione vi risparmierà ore di debug, quando non riuscirete a capire perché quel codice "così ovvio" funzioni perfettamente sulla piattaforma X, ma sbagli *inspiegabilmente* su quella Y.

2) VOID MAIN()

Quest'errore è una specie di marchio d'infamia; è considerato universalmente: "il biglietto da visita dell'ignorante in C++".
Il fatto che il vostro compilatore accetti scritture come *void main(void)*, o *void main()*, o *main()* senza obiettare il minimo warning, non significa che queste siano corrette. (Ma probabilmente significa che dovreste pensare seriamente a cambiare compilatore).
Neanche il fatto che il vostro manuale abbia ripetuto dalla prima all'ultima pagina *void main(void)*, significa che questa scrittura sia corretta. (Ma vi spiega perché gli esperti vi scongiurano di buttare via i libri di Schildt, per esempio).
Immagino che la conclusione di questo preambolo sia chiara, alla luce del paragrafo precedente: l'unica autorità col potere di decidere cosa sia universalmente giusto, è lo standard. **E *void main()* <u>non</u> lo è**.
Secondo lo standard ISO C++ (vedi 3.6.1[2]), infatti, le **uniche** due scritture universalmente ammesse sono:

```
int main()
```

e

```
int main(int argc, char* argv[])
```

La costante è che in entrambi i casi il tipo restituito *dev'essere* **int**. E in effetti questo ha un senso, perché nei sistemi POSIX alla fine dell'esecuzione i programmi restituiscono sempre un intero al chiamante. A tal proposito, questo:

```
#include <iostream>

int main() {
        std::cout << "Io sono standard!\n";
}
```

è un programma corretto secondo lo standard, anche se manca l'istruzione return. In questo caso, infatti, viene restituito implicitamente in valore 0 (che nei sistemi POSIX significa: "tutto è andato bene").
Già che ci siamo, la scrittura:

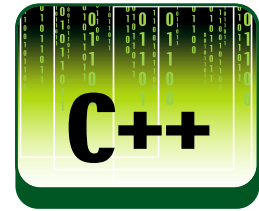
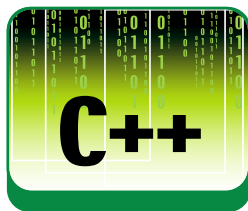
```
main()
```

Non è corretta secondo il C++ standard (mentre lo è secondo il C '89), perché il C++ non assume implicitamente int per le funzioni che non presentano un tipo di ritorno.
Voglio ribadire un concetto importante: i compilatori hanno tutto il diritto di permettere **altre** scritture di main. Possono in linea teorica esistere compilatori che permettono *long main()*, o *char* main()*, e sicuramente anche *void main()*, dato che possono esserci casi in cui in cui il sistema… semplicemente non esiste.
Il problema è che tutte queste sono eccezioni, estensioni del linguaggio: **pochi compilatori accetteranno scritture come *void main()*, ma tutti *sono obbligati* ad accettare *int main()***. È questa la forza e la sicurezza di scrivere in C++ standard (oltre a non farvi prendere in giro da gente su Internet che vive per divertirsi alle spalle dell'ennesimo novellino che osi scrivere *void main(void)*).

3) #INCLUDE <IOSTREAM.H>

Quest'errore appartiene al regno del "*codice che era valido una volta, ma da allora sono passati almeno dieci anni*". Ecco un esempio:

```
//nota: questo codice è (come minimo) deprecato
#include <iostream.h>

int main() {
        cout << "Ciao, mondo!" << endl;
}
```

Qualunque libro o tutorial vi presenti oggi un "*hello world*" del genere è molto vecchio, o molto scadente (in ogni caso, passate ad altro).

I file della libreria standard che terminavano con ".h" sono stati deprecati – ed è successo più di dieci anni fa!
Gli header "moderni" non hanno alcun'estensione (come <iostream>), e presentano molti vantaggi rispetto ai vecchi: utilizzano il namespace **std**, sono fortemente orientati ai template e si integrano molto meglio col resto della libreria standard. L'equivalente moderno del programma qui sopra è:

```
#include <iostream>

int main() {
        std::cout << "Ciao, mondo!" <<
                                 std::endl;
}
```

Nel nuovo codice è obbligatorio qualificare i nomi della libreria standard, usando il prefisso **std::**. Se non volete allungare eccessivamente il codice, potete usare la direttiva **using namespace**:

```
#include <iostream>

int main() {
        using namespace std;
        cout << "Ciao, mondo!" << endl;
}
```

Molti programmatori tendono però ad abusare di questa direttiva, annullando così tutti i vantaggi dei namespace (cioè di evitare collisioni fra i nomi che create voi e quelli che usa la libreria standard). Pertanto, cercate di usarla solo se i nomi standard vengono usati tanto estensivamente che gli std:: complicano *davvero* la lettura. E cercate sempre di posizionarla quanto più internamente possibile. Una cattiva pratica molto diffusa, ad esempio, è definire un *using namespace* globale:

```
#include <iostream>

using namespace std; //perché all'esterno?

int main() {
        cout << "Ciao, mondo!" << endl;
}
```

Purtroppo al mondo c'è un numero sorprendente di programmatori che non si interessano né dello standard, né tantomeno dello stile (sono quelli del tipo: "compila quindi è giusto!"). Così, ad un certo punto, gli scrittori di compilatori hanno capito che l'unica strada per far smettere di usare <iostream.h>, consisteva nel togliere i file ".h" dalle loro implementazioni della libreria standard.
Questa è probabilmente la ragione definitiva che dovrebbe spingervi ad evitare l'inclusione di file di libreria ".h": sulla maggior parte dei compilatori più importanti, **semplicemente non esistono**.

4) #DEFINE PI 3.1415

Questo ricade in quella (vasta) categoria di errori commessi da tutti i neofiti che pensano che il C e il C++ siano "più o meno la stessa cosa". È vero che il C++ è (quasi) un soprainsieme del C, ma le funzionalità aggiunte sono talmente tante da cambiare radicalmente le pratiche di programmazione. Non di rado capita che ciò che è caldamente raccomandato in C, sia fortemente sconsigliato in C++ (e viceversa).
Usare #define per definire delle costanti, ad esempio, è tipico in C, perché la parola chiave *const* non vincola sufficientemente il programmatore. In C++, invece, **const** è una cosa seria: un vincolo che è impossibile aggirare, se non per mezzo di orrendi const_cast.
Usare #define non permette al compilatore di inserire la costante all'interno della tabella dei simboli, perché all'unità di traduzione tale costante non arriva nemmeno: viene intercettata e sostituita dal preprocessore. In questo modo ottimizzazioni, controlli, messaggi di errore significativi, diventano tutti sogni impossibili.
Un programmatore C++ usa const ogni volta che può, in tutte le sue accezioni possibili:

```
//definizione di costante (e non semplice macro)
const Pi = 3.1415; //non #define PI 3.14592

//passaggio per riferimento costante
//(usate la copia soltanto per tipi semplici o
                                    primitivi)

void Funzione(const Oggetto&); //non void
                           Funzione(Oggetto)

//funzione membro costante
void Oggetto::Funzione() const {}
```

Un errore ancora peggiore è l'uso del preprocessore per definire **funzioni**, come questa:

```
#define ABS(n) n > 0 ? n : -n
```

A parte il tipico errore di omettere le parentesi attorno all'argomento (n), questo tipo di macro è un generatore perpetuo di sorprese e

bug critici. Se non ci credete, provate a scrivere:

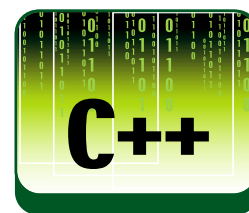
```
#define ABS(n) (n) > 0 ? (n) : -(n)

int main() {
        int n=0;
        std::cout << ABS(n--);
}
```

Qual è l'output del programma? Sorprese come questa vi spiegano perché in generale gli esperti di C++ vedano di pessimo occhio l'uso del preprocessore (se non per il controllo condizionale della compilazione con #ifdef). In casi come questo C++ ha, peraltro, un'alternativa migliore sotto ogni aspetto: le funzioni **template inline**.

```
template<typename T>
inline T Abs(T n) {
    return n > 0 ? n : -n;
}

int main() {
        std::cout << Abs(-10);

}
```

Poiché la funzione è template, è valida per ogni tipo (anche non primitivo, volendo), proprio come la macro. Poiché è inline, verrà espansa nel codice, proprio come la macro. Ma a differenza della macro, sarà una **funzione**, sotto stretto controllo del compilatore.

5) CHAR* STRINGA

Quest'errore ricade nella stessa categoria del precedente. Mentre in C gli array di char sono comunissimi, nel mondo C++ le stringhe primitive sono viste come una delle principali fonti di bug, e vengono usate solo in casi in cui le prestazioni non permettono altro (e sono casi rari).
Assieme alle creme antibrufoli, gli array di char sono il paradiso dei cracker: dei buffer a lunghezza fissa terminati dal carattere nullo. Sembrano inventati apposta per permettere il prossimo exploit da stack-based overflow!
Il C++ offre al programmatore il tipo **std::string** (o *basic_string<T>*, se preferite un approccio più generico), che evita ogni genere di fastidio permettendo operazioni di copia, assegnamento, ricerca, sostituzione, iterazione, e mille altre meraviglie, sotto forma di semplici funzioni membro. Oltre a queste, potete mettere nel computo anche tutti gli algoritmi STL di <algorithm> e l'integrazione con le altre componenti della libreria standard (come stringstream, ad esempio).
Questo errore è soltanto un caso particolare. In generale, evitate di usare (e soprattutto passare in giro) degli array: preferite loro dei **vector** (o se proprio volete la lunghezza fissa, provate con **boost::array**). Non usate <cstdio>, usate gli stream. **Non usate il C, usate il C++** (e questo ***comprende*** la libreria standard).

6) VOID* FUNZIONE (INT NARGS, ...)

Il C++ è un linguaggio tipizzato fortemente e staticamente, e l'attenzione per il type-checking è molto maggiore rispetto a quanto avviene in C. I controlli effettuati dai compilatori C++ sono impietosi, ed evitano che a runtime succedano "cose strane", come ad esempio che un intero venga interpretato come se fosse una stringa, mandando in crash il programma (nel migliore dei casi).
Anche in C esistono controlli simili ma, come abbiamo visto nel paragrafo sul qualificatore *const*, sono molto meno rigidi.
In C è comune, ad esempio, usare funzioni che operano per **ellissi**, ovverosia con parametri di numero e tipo variabile. La funzione ***printf*** ne è l'esempio principe:

```
#include <cstdio> //e non <stdio.h>!
int main() {
std::printf("%s %s!", ciao, mondo);
}
```

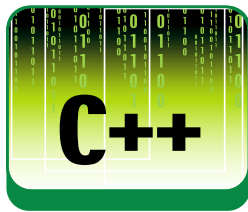
Quale sarà il risultato dell'esecuzione? Dipende dal valore e soprattutto dal **tipo** delle variabili *ciao* e *mondo*. Nel caso seguente, l'esecuzione andrà a buon fine:

```
const char* ciao = "Ciao";
const char* mondo = "mondo";
```

Ma in quest'altro, proprio no.

```
int ciao = 1;
char mondo = 2;
```

Possiamo solo immaginare cosa succederà in un caso simile. Probabilmente un crash dell'applicazione, o un mare di scritte strane, o entrambe le cose. Stesso discorso per quest'istruzione:

```
    std::printf("%s %s!", "Ciao, Mondo!");
```

In questo caso il tipo dei parametri è giusto ma è il loro *numero* ad essere sbagliato (ne manca uno!). Altro crash a tempo di esecuzione, impossibile da prevedere per un compilatore.
È per questo che il C++ ha un sistema di **stream**, perfettamente sicuro rispetto ai tipi. Se proprio non vi piace o avete bisogno di una stringa di formattazione, potete usare strumenti C++ sicuri, che traggano vantaggio dall'overloading degli operatori, come il **boost::format** che abbiamo visto su questa stessa rubrica nell'Aprile 2007.
Ma l'ellissi non è che una delle operazioni volte ad aggirare pericolosamente i controlli del compilatore. Anche alcuni esempi di upcasting su tipi non-polimorfi sottraggono irrimediabilmente informazioni al compilatore, e va ancora peggio con alcuni esempi di type punning che vengono visti male persino in C, come questi:

```
union Variant {
        int intero;
        char* stringa;
};

void* tipoGenerico = reinterpret_cast<void*>(x);
```

Entrambi i casi rendono impotente il compilatore di fronte a vostri eventuali errori. Quando usate la union ci si può solo fidare che stiate leggendo dal campo corretto, e quando ripristinate il casting da void*, c'è solo da pregare che azzecchiate il tipo originale di x, qualunque esso fosse.
Morale della favola: in C++ il compilatore è il vostro migliore amico. Prenderlo in giro è una gran brutta azione, e spesso si paga.

7) RETURN NEW INT[200]

La gestione della memoria è uno dei punti critici del C++. Vi è concesso controllare fino all'ultimo bit, gestire allocazioni e deallocazioni direttamente e in maniera più furba ed efficiente di come potrebbe fare un garbage collector. Ma vi è anche concesso di fare grandi danni, e di complicare la vostra applicazione al punto da non riuscire più a capire se e quando una risorsa sia stata allocata, o distrutta.
Per questo la gestione diretta della memoria (e delle risorse!) è quasi sempre un male. Ogni volta che scrivete un'istruzione ***new***, vi consiglio di sollevare un sopracciglio.
Innanzitutto assicuratevi che esista una corrispettiva istruzione delete.

```
new x
... istruzioni intermedie ...
delete x
```

Poi assicuratevi che fra il new e il delete non ci siano punti di uscita, **anche impliciti**. Se una delle istruzioni intermedie è in grado di lanciare un'eccezione, ad esempio, x sarà a rischio di memory leak. E, ancora peggio, se x detiene delle risorse (un file, un hook, una qualunque risorsa di sistema), queste non verranno mai rilasciate, perché il distruttore non sarà mai eseguito.
In C++ si usa la tecnica RAII per la gestione di questi problemi. Abbiamo dedicato ben quattro puntate di questa rubrica ad analizzarle (dal Dicembre 2006 al Marzo 2007).
Potete creare una classe RAII, potete usare gli smart pointer, potete usare uno ScopeGuard come quello fornito da Loki (e che vedremo in uno dei prossimi appuntamenti), potete addirittura utilizzare un garbage collector, ma non usate mai la coppia **new/delete** da sola, senza l'analisi delle istruzioni intermedie e dei punti di uscita.
Per quanto riguarda le funzioni, se proprio dovete crearne una che restituisca un puntatore a memoria dinamica (tipicamente per preservare il polimorfismo), assicuratevi quantomeno di incapsularla in un auto_ptr.

```
struct Mostro {};
struct Zombie : Mostro {};
struct Mostriciattolo : Mostro {};

std::auto_ptr<Mostro> CreaMostro(const
                                 std::string& tipo)

{
  if (tipo == "Zombie")
    return std::auto_ptr<Mostro>(new Zombie);
  else
    return std::auto_ptr<Mostro>(new
                                 Mostriciattolo);
}
```

In questo modo se il chiamante ignorerà il risultato di *CreaMostro*, questo sarà distrutto automaticamente. E se, invece, il chiamante vorrà usare il mostro, sarà obbligato a gestirlo a sua volta attraverso uno smart pointer.
Per quanto riguarda le classi, non restituite in nessun caso riferimenti o puntatori diretti alle risorse che queste detengono:

```
class Bitmap {
        char* rawData_; //Dati dei Pixel
public:
```

```
        char* GetRawData() { //NO!
return rawData_;
};

//... altre funzioni membro
};
```

Nel caso giocattolo presentato qui sopra, la classe Bitmap detiene un buffer allocato dinamicamente (rawData_), contenente il valore di ogni singolo pixel, ed espone una serie di funzioni membro (come SetPixel(x, y, colore), eccetera…), per modificarlo. E, ovviamente, dealloca il buffer nel suo distruttore.
Tutto questo assicura che, se la classe Bitmap è ben fatta, non potranno esserci dei leak di memoria o di risorse. Se, però, create una funzione membro come ***GetRawData***, che espone direttamente il buffer, avete vanificato tutto in un colpo solo. L'utente potrà giocarci a suo piacimento, deallocarlo, corromperlo, il tutto in barba alla classe Bitmap che non avrà più alcun controllo su risorse che dovrebbero essere *soltanto sue*.

8) CLASS O STRUCT?

Il finale del paragrafo precedente è in realtà solo un caso particolare di un errore più generico e molto frequente nei principianti: confondere classi e strutture.
Cominciamo a sfatare un luogo comune molto diffuso, e cioè che la parola chiave *struct* sia "meno potente" di *class*. Le strutture permettono di definire costruttori, distruttori, funzioni membro (anche virtuali), liste di tipi da cui ereditare, operatori da sovraccaricare, eccetera… esattamente come le classi.
Allora qual è la differenza? Da un punto di vista formale, *l'**unica*** differenza è che nell'elenco delle superclassi e dei membri di una struttura viene sottintesa la parola chiave *public*, anziché *private*.

```
class Zombie : Mostro //Ereditarietà privata
{
        Zombie(); //Costruttore privato!
};

struct Zombie : Mostro //Ereditarietà pubblica
{
        Zombie(); //Costruttore pubblico
};
```

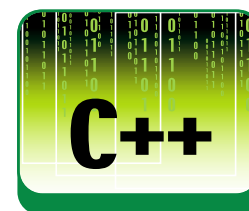
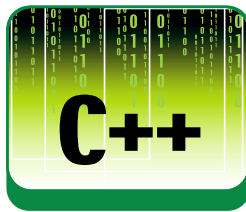
Questa piccola differenza sintattica, riflette, però un'enorme differenza a livello concettuale. Le classi devono **nascondere**, mentre le struct normalmente si preoccupano solo di mostrare.
Una struct consiste solitamente in un insieme di dati membro pubblici, con l'aggiunta (talvolta) di qualche operatore o funzione membro pubblica. Un utente può tranquillamente modificarne ogni campo, senza paura di comprometterne l'integrità:

```
struct Punto {
        int x;
        int y;
};
```

Le classi, invece, mantengono al loro interno qualche risorsa segreta o impossibile da modificare direttamente. Tutte le funzioni membro servono ad accedervi in modo controllato. Ecco un esempio:

```
class Frazione {
        int numeratore_;
        int denominatore_;
        void Semplifica();
public:
        Frazione(int, int);
        int Numeratore();
        int Denominatore();
};
```

La classe Frazione ha un numeratore e denominatore privati. L'accesso viene fornito (in sola lettura, in questo caso) soltanto attraverso delle funzioni membro pubbliche. Notate che la classe ha anche una funzione membro per semplificare la frazione, ma questa non è visibile all'esterno, per il semplice fatto che **non deve interessare all'utente**. È la classe stessa a preoccuparsi che *Semplifica* venga richiamata automaticamente ad ogni modifica.
In questo modo, si garantisce che non potrà esistere in nessun modo e in nessun momento, una Frazione non semplificata. Questa condizione garantita prende il nome di **invariante**.
Frazione, quindi, **dev'essere definita come class** perché deve preservare l'invariante di essere sempre in uno stato in cui il numeratore e il denominatore non hanno fattori in comune.
Ma quando non c'è alcun'invariante da mantenere, è inutile usare una classe. E invece, capita molto spesso di vedere qualcosa di simile:

```
class Punto {
        int x_;
        int y_;
public:
        int GetX() {return x_;}
```

```
    int GetY() {return y_;}
    void SetX(int x) {x_ = x;}
    void SetY(int y) {y_ = y;}
};
```

La classe Punto non preserva alcun invariante: è soltanto una serie di dati nascosti e di funzioni membro pubbliche che si limitano a restituirli (**getter**) o ad assegnarli direttamente (**setter**). Si tratta, in sostanza, della stessa *struct Punto* che abbiamo mostrato poco fa.
Chiamare classe un'accozzaglia di getter e setter non rende il design più OOP: è soltanto un'inutile complicazione. In casi come questo, è meglio definire Punto come una **struct**: si risparmiano linee di codice, si offre una maggiore comodità all'utente, e soprattutto non lo si induce a credere che la classe *Punto* sia più intelligente di quanto è in realtà.
I casi in cui è legittimo che un tipo non preservi uno straccio di invariante sono molto rari. Se il vostro codice espone diversi tipi sotto forma di struct (o, ancor peggio, di strutture mascherate da classi), cominciate a chiedervi cosa non va nell'architettura della vostra applicazione.

9) I++

Fra tutti gli errori riportati in quest'articolo, questo è senz'altro il meno grave, ma si guadagna il posto in graduatoria per la sua eccezionale diffusione. Non è raro trovarlo anche in codice di qualità e persino su insospettabili libri di testo.
In C++ esistono due operatori di incremento, uno postfisso (i++) e uno prefisso (++i). Visti come istruzioni, sono identici: entrambi significano "*i = i + 1*". Visti come espressioni, però, differiscono radicalmente: *i*++ vale i, mentre ++*i* vale i+1.
Se il vostro grado di competenza va da "princi-

PER APPROFONDIRE SULLA CARTA

Se il leggero antipasto che vi ho presentato con questo decalogo vi è piaciuto, o se vi aspettavate un menù più consistente, forse è il caso che diate un'occhiata alla vasta letteratura in materia. Vista la complessità del C++ e la serie di trappole in cui possono cadere i programmatori, in effetti, non c'è da stupirsi se più di un libro è stato scritto appositamente per chiarire *cosa non fare in* C++.

***C++ Gotchas*, di Stephen C. Dewhurst, è probabilmente il più noto e completo. Si tratta 99 consigli su pratiche da non seguire. In ogni tema si discute del perché un certo idioma nasconda una trappola, e le varie soluzioni alternative che dovrebbero invece essere prese in considerazione. Caldamente consigliato.**

***How Not To Program In C++*, di Steve Oualline, è un libro molto più leggero del precedente. Scritto in uno stile molto ironico e piacevole, propone 111 programmi "non funzionanti", intervallati da storielle, metafore e barzellette IT. Non offre una gran profondità, ma può essere una lettura simpatica, e un ottima fonte per quei docenti che si divertono a far ragionare i propri allievi.**

***Imperfect C++*, di Matthew Wilson, è un libro molto particolare, dal momento che discute più che altro del perché il C++ non possa considerarsi un linguaggio perfetto. Con questa scusa, in realtà si danno molti buoni consigli, in uno stile molto chiaro e colloquiale. A dir la verità qualche consiglio è discutibile, e lo stile, molto spesso, suona anche un po' arrogante. Ma è molto facile sembrare odiosi scrivendo libri del genere (come probabilmente dimostra questo stesso articolo).**

Tutti questi testi possono essere presi come accessori, e ognuno offre qualcosa di diverso. Le guide più importanti, tuttavia, sono i cosiddetti "secondi libri", quelli che devono essere studiati dopo il manuale introduttivo e devono necessariamente figurare sugli scaffali di qualunque programmatore C++.

***Effective C++*, di Scott Meyers: 55 temi su pratiche da seguire e da evitare. È universalmente noto come il "primo fra i secondi libri". Se avete appena finito di leggere il vostro primo manuale, ci metterete molto tempo a digerirlo, e vi consiglio di tenerlo per qualche mese sulla scrivania mentre programmate.**

***More Effective C++*, altri 35 temi. Si tratta di un libro meno fondamentale del precedente, tuttavia è un'ottima appendice. Alcune sezioni sull'implementazione di smart pointer e dei sistemi di copy on write sono decisamente chiare e interessanti.**

***Effective STL*, è l'equivalente di Effective C++ per la libreria standard. Dal momento che i manuali solitamente danno qualche consiglio pratico sul linguaggio, ma trattano poco l'STL, è forse il più fondamentale dei tre Effective qui presentati. Se il C++ è insidioso, la sua libreria standard è ancora peggio. Indispensabile.**

***C++ Coding Standards*, di Herb Sutter e Andrei Alexandrescu. Cosa si intende esattamente per "stile"? Le infinite diatribe su come gestire le indentazioni e il casing delle variabili? Oppure queste guerre di religione mascherano soltanto i veri problemi? Questo testo è un'ottima risposta a tali domande: presenta 101 linee guida per lo stile su cui è difficile non essere d'accordo. Molto semplice da seguire anche per il neofita, questo è sicuramente un libro da leggere parallelamente al vostro primo Effective.**

***Exceptional C++*, *More Exceptional C++* e *Exceptional C++ Style*, di Herb Sutter. Se avete letto tutti i libri precedenti, siete pronti per affrontare questi tre. Si tratta di temi dalla forma di quiz: si illustra un problema, si chiede come lo risolvereste voi, e poi si analizzano le varie soluzioni possibili dando un voto a ciascuna. Sutter è contemporaneamente uno dei migliori esperti di C++ e uno dei migliori divulgatori, e fra loro è forse l'unico in grado di spiegare con estrema competenza e chiarezza come certe funzionalità del C++ possono essere implementate dai vari compilatori.**

piante" a "mediamente esperto", probabilmente a questo punto starete pensando: "*E allora? Dimmi qualcosa che non so.*" (Se invece siete *davvero* esperti, avrete già capito dove voglio andare a parare. Potete evitare di leggere il resto del paragrafo).
Il problema reale è che molti programmatori hanno un ingiustificato e morboso attaccamento all'operatore postfisso. Prendiamo, ad esempio, il tipico programma per stampare i numeri da 0 a 99. Su molti manuali, troverete qualcosa del genere:

```
#include <iostream>

int main() {
    //nota: questo ciclo è perfettibile
    for (int i=0; i<100; i++) {
        std::cout << i << std::endl;
    }
}
```

Ma siete sicuri che "i++" sia proprio l'operatore più indicato? In realtà in questo caso l'operazione di incremento non viene usata come espressione, ma semplicemente come istruzione. Quindi l'operatore più corretto è quello **prefisso:**

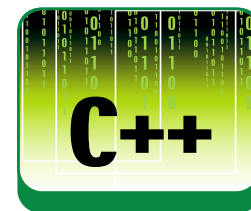
```
#include <iostream>

int main() {
    for (int i=0; i<100; ++i) { //OK!

        std::cout << i << std::endl;
    }
}
```

Non si tratta di una semplice questione stilistica: c'è un vantaggio effettivo che si capisce solo se si conosce come funziona l'operatore postfisso. Prima che la variabile venga incrementata, viene creata una sua **copia**, che viene quindi usata come valore dell'espressione.
Pertanto, se scrivete "i++" come incremento del ciclo, fate sprecare al programma 100 operazioni di copia per un valore che non interessa a nessuno. A volte i compilatori riescono a capire l'errore e lo correggono, ma spesso non hanno i mezzi (o non sono abbastanza furbi) per farlo. In ogni caso, perché rischiare, quando potete scrivere semplicemente e molto più correttamente ++i?
La regola è semplice: se dovete usare una variabile "i" come espressione e contemporaneamente incrementarla, scrivete **i**++. Altrimenti (cioè nella maggior parte dei casi, compresi i cicli) usate ++**i**.

10) "IO QUELLA ROBA NON LA USO"

Quello che segue è uno stralcio della "Mozilla Compatibility Guide", che potete trovare all'indirizzo http://www.mozilla.org/hacking/portable-cpp. html.

- Non usate i template
- Non usate le eccezioni
- Non usate l'RTTI
- Non usate la libreria standard del C++, incluso iostream
- Non usate i namespace

L'intera lista è lunga, ma si traduce più facilmente in: non usate il C++. Il linguaggio risultante da questa serie di menomazioni è un sottoinsieme penoso del C++, al cui confronto perfino il GWBASIC splenderebbe di un'aura di dignità accademica.
Liste come questa sono nate più di dieci anni fa, quando ancora il C++ si identificava con CFront e i compilatori faticavano ad imitarne il comportamento, privi di una vera e propria standardizzazione. In questa situazione i progetti che miravano ad una certa compatibilità decisero di imporsi pesanti limitazioni, pur di facilitare le operazioni di porting. Il C++ veniva visto, più che altro, come un C *on steroids*.
Oggi tutto ciò non ha più alcun senso. Un compilatore che fatichi ad implementare caratteristiche standardizzate da dieci anni, come namespace, libreria standard ed eccezioni è un compilatore scadente, e basta. Per "migliorare la compatibilità" è sufficiente buttarlo via e prenderne uno decente.
Nel C++ moderno il semplice atteggiamento dell'"io uso solo le classi, il resto non mi interessa" è insostenibile. In particolare, è impossibile fare a meno di funzionalità come template ed eccezioni: ogni volta che userete la libreria standard, o le principali estensioni come il Tr1, Boost e Loki, le userete implicitamente.
Vi consiglio, in particolare, di spendere molto tempo ad imparare ad usare i template, in tutte le loro sfumature, compresa la specializzazione parziale e qualche base di metaprogrammazione: scriverete codice molto più riutilizzabile e compatto. (Se volete qualche esempio, provate a confrontare il codice di Smartwin++ con quello di colossi "vecchio stile", come QT e wxWidgets).
E se proprio volete seguire una lista di cose da non fare in C++, questo articolo ve ne offre già dieci…

Roberto Allegra

L'AUTORE

Il sito www.robertoallegra.it **contiene l'elenco degli articoli pubblicati in questa rubrica, con gli inevitabili approfondimenti ed errata corrige degli articoli e del codice. L'e-mail dell'autore è** posta@robertoallegra.it.

IL RITORNO DI PACKMAN PER POCKET

SICURAMENTE LA MAGGIOR PARTE DI VOI RICORDANO PACKMAN: IL PUPAZZO SFERICO CHE DEVE MANGIARE QUANTE PIÙ PILLOLE POSSIBILI SENZA FARSI CATTURARE DAI FANTASMI DI TURNO. IN QUESTO ARTICOLO NE PROGRAMMEREMO UNO PER CELLULARE

Nonostante la costruzione di applicazioni per smart device debba tenere conto delle limitatezza delle risorse di sistema è comunque possibile programmare un videogame sufficientemente professionale e accattivante. In particolare il gioco sarà costituito dai seguenti elementi:

1. un Packman animato orientabile
2. un campo da gioco costituito da un certo numero di puntini colorati (le pillole)
3. quattro pulsanti software ed altrettanti hardware che consentono di spostare Packman
4. due punteggi: il primo che si aggiorna mano a mano che vengono mangiate le pillole, il secondo quando un fantasma tocca Packman
5. un numero via via crescente (a seconda del livello) di fantasmi che cercano di catturare Packman
6. piú livelli con un crescente grado di difficoltá
7. effetti sonori.

Fig. 1: L'applicazione originale Pacman di Atari.

PARTICOLARITÀ DEL FRONT-END

Una volta che avete creato un *New Project* per Windows Mobile 5 per Pocket PC, visualizzate la Toolbox e inserite i componenti grafici richiesti. In particolare un controllo *Panel* per il campo da gioco e alcune *PictureBox*: una sarà Packman, le altre quattro i terribili fantasmi che renderanno la vita difficile al protagonista del videogame. Per animare Packman è necessario introdurre una *ImageList* per ogni posizione che esso deve assumere, ad esempio battezzatele come segue: *imaAnimaSx*, *imaAnimaSu*, *imaAnimaDx* e *imaAnimaGiu*. Ognuna deve contenere le immagini che trovate nella directory PackMan\Immagini del CD: le sx(n), su(n), dx(n) e giu(n). Caricatele in ciascuna proprietà Images facendo clic sul pulsante *Add* a partire

Fig. 2: Ecco, invece, Packman mentre viene eseguita all'interno dell'emulatore per Pocket PC WM 5.0.

dalla prima (ad es. sx1, poi sx2 ed infine sx3). Siccome dovete animare anche i fantasmini avete bisogno di un oggetto *ImageList* per ciascuno di essi: *imaMostro1*, *imaMostro2*, *imaMostro3* e *imaMostro4*. Anche in questo caso selezionate le immagini che trovate nel CD: le fantaAra(n), fantaRo(n), fantaVer(n) e fantaVio(n). Questa volta fate attenzione a caricarle in maniera non consecutiva nelle proprietà *Images*: fate clic sul pulsante Add a partire dalla prima per *picMostro1* e *picMostro3*, mentre per *picMostro2* e *picMostro4* partite dalla seconda fino ad arrivare alla prima (ad esempio la sequenza per *picMostro2* sarà: fantaAra2, fantaAra3, fantaAra4 e fantaAra1). In questo modo garantirete un'asincronia di movimenti delle varie animazioni.

Fig. 3: Per iniziare una nuova partita bisogna agire sul menu File.

IL CAMPO DA GIOCO

La creazione del campo da gioco consiste nella definizione delle pillole colorate a disposizione del vorace Packman. La prima opzione che viene in mente per creare le pillole è quella di utilizzare un controllo come la *Label* o il *Panel*, ma deve essere scartata subito! Il perché è semplice: la definizione di cosí tanti controlli (un centinaio) renderebbe estremamente lenta l'applicazione. Il modo più semplice e compatibile con ottime performance del Pocket PC è quello di utilizzare il metodo *DrawEllipse* della classe *Graphics* di Visual Basic.NET.
Per procedere in questo modo dovete prima di tutto creare due variabili globali: una di tipo bitmap (*Dim Immagine as bitmap*) e l'altra graphics (*Dim Grafici as graphics*). Successivamente dovete inserire all'interno dell'evento *Load* della form le righe riportate di seguito: le prime due sono necessarie (se abbinate alle istruzioni inserite nell'evento *Paint* di *panSfondo*) per evitare che i pallini scompaiano se viene ceduto il controllo ad un altro programma, mentre l'ultima serve a creare una bitmap con sfondo bianco.

```
Private Sub PackMan_Load(...) Handles MyBase.Load
  Me.Immagine = New
                        Bitmap(Me.panSfondo.Size.Width,
                             Me.panSfondo.Size.Height)

  Me.Grafici = Graphics.FromImage(Me.Immagine)
  Me.Grafici.FillRectangle(New
          SolidBrush(Color.White),0,0,Me.Size.Width,
                                      Me.Size.Height)

  pMettiPallini()
End Sub
```

La procedura *pMettiPallini()* richiamata anch'essa nell'evento *Load* della form serve a creare e a posizionare le pillole colorate sul campo da gioco.

```
Private Sub pMettiPallini()
    Dim i, riga, col As Integer
    Dim myPen As Pen
    Cursor.Current = Cursors.WaitCursor
    For col = 0 To 200 Step 20
        For riga = 1 To 200 Step 20
            myPen = fColoraPalliniRandom(myPen)
            For i = 0 To 20
                Me.Grafici.DrawEllipse(myPen, (10 +
                                   col) + i / 2, _
                    (10 + riga) + i / 2, 10 - i, 10 - i)
            Next
        Next
    Next
    Me.panSfondo.Invalidate()
    myPen.Dispose()
    Cursor.Current = Cursors.Default
End Sub
```

Fig. 4: Packman a design time.

LE REGION

Creare Region serve a rendere piú ordinato e leggibile il codice, riducendo i ben noti problemi di manutenzione. La sintassi è #Region "Nome regione"... #End Region. Le sezioni di codice riportate al loro interno vengono compresse e, quindi, nascoste. Si consiglia di comprendere in una Region gli algoritmi che hanno logica simile oppure che svolgono azioni simili (ad es. variabili, eventi, procedure).

Come potete vedere sono stati utilizzati tre cicli *For...Next* annidati l'uno nell'altro. Cominciamo dal più interno. Esso serve per disegnare 20 ellissi, al fine di creare un cerchio completamente colorato. Come anticipato dovete utilizzare il metodo *DrawEllipse* della classe *Graphics* ed è necessario definire la penna per poter disegnare le ellissi. Questo metodo grafico ha bisogno dei seguenti parametri:

1. penna colorata *myPen*
2. coordinate X e Y per posizionare l'oggetto sulla form
3. dimensioni dell'oggetto

Ad esempio per creare un pallino rosso senza riempimento, posizionato in ascissa 10, ordinata 12 e con dimensioni pari a 20 per 20 pixel è sufficiente codificare le due righe che seguono.

```
myPen = New Pen(Red)
Me.Grafici.DrawEllipse(myPen, 10, 12, 20, 20)
```

Fig. 5: La finestra di dialogo che permette di inserire nuove immagini nel controllo ImageList.

NOTA

L'EVENTO MNUNUOVO_CLICK

Questo evento serve a riinizializzare tutte le variabili ogni volta che si decide di incominciare una nuova partita. In particolare la ReDim arRigaCol(183, 201) azzera il contenuto di tutte le celle della matrice, ponendole uguali a stringa vuota.

Per riempire il pallino dovete appunto ciclare una ventina di volte riducendo via via il cerchio, ossia cambiando ogni volta le sue coordinate: questo obiettivo si può ottenere facilmente aggiungendo alle coordinate X e Y che consentono di posizionare il primo oggetto (10 e 10), rispettivamente il valore della colonna e della riga corrente, modificati dai cicli precedenti e cioè: *For col = 0 To 200 Step 20 e For riga = 1 To 200 Step 20*. Notare che quando ciclate sulla variabile col effettuate non 200 loop, ma solo 11 perché è stata impostata la clausola Step che impone di procedere di venti in venti. Nel caso di riga otterrete, invece, 10 loop perché l'indice di partenza è 1. Non è un caso, ovviamente, che le colonne siano 11 e le righe 10.

In questo algoritmo è stata introdotta volutamente una finezza la colorazione dei pallini in maniera casuale grazie alla funzione fColoraPalliniRandom riportata di seguito.

```
Private Function fColoraPalliniRandom(ByVal
                                   myPen As Pen)
    Randomize()
    myPen = New
        Pen(Color.FromArgb(m_Random.Next(255),
                           m_Random.Next(255), _
        m_Random.Next(255)))
    fColoraPalliniRandom = myPen
End Function
```

La prima istruzione *Randomize()* consente d'inizializzare il generatore dei numeri casuali. Senza *Randomize()* le chiamate a *Rnd()* genererebbero sempre la stessa sequenza di numeri e, quindi, di colori. La funzione *Rnd* viene, poi, utilizzata per generare i numeri random che applicati al metodo *Color.FromArgb* ci danno il colore casuale.

Ovviamente *fColoraPalliniRandom* è una funzione non di VB.NET, ma creata da voi. A questo proposito è opportuno ricordare quando si utilizzano le procedure (Sub) e quando le funzioni. Entrambe sono in grado di ricevere uno o piú parametri in input, ma solo le Function possono restituire valori. In questo caso era necessario restituire alla procedura chiamante la penna colorata *myPen*.

Infine l'istruzione Invalidate forza il disegno dell'immagine sul pannello *panSfondo*, mentre la Dispose rilascia tutte le risorse grafiche utilizzate dall'oggetto *myPen*, liberando preziosa memoria per il Pocket PC.

Per concludere questo paragrafo facciamo un rapido riferimento alla gestione dei cursori del mouse. In situazioni in cui esistono momenti di attesa dovuti ad algoritmi particolarmente pesanti è importante che visualizzate l'icona di attesa (corrispondente alla clessidra in ambiente PC). Prima di tali algoritmi dovete uguagliare la proprietà *Current* dell'oggetto *Cursor* ad un valore differente da quello predefinito, cioè a *Cursors.WaitCursor*. In questo modo il cursore correntemente visualizzato viene modificato e l'applicazione smette di attendere eventi del

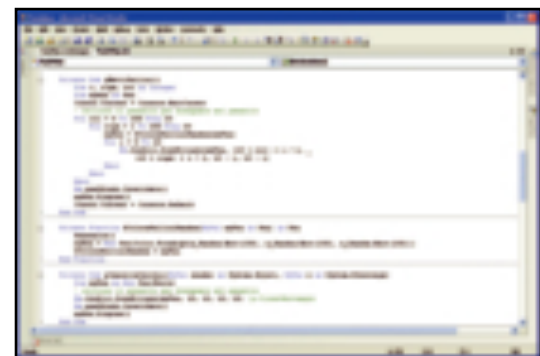



Fig. 6: Le funzioni che gestiscono la creazione delle "pillole", il loro colore casuale e la loro cancellazione.

mouse; ciò significa che qualsiasi azione effettuata sul monitor del Pocket PC o sui tasti hardware non ha alcun effetto. Al termine dell'elaborazione non dimenticatevi di reimpostare i valori di default con *Cursor.Current = Cursors.Default*.

ANIMARE PACKMAN

Inserite nel progetto un controllo *Timer*, caratterizzato dalle seguenti proprietà:

- *(Name)= tmrAnima*
- *Enabled=True* (in questo modo il Timer risulta attivo in fase di esecuzione)
- *Interval=150* (frequenza in millisecondi di attivazione dell'evento Tick)

Per rendere più avvincente il gioco dovete fare in modo che Packman apra e chiuda la bocca e che orienti sempre il volto nella direzione verso cui si sta muovendo.

Per animare Packman dovete inserire nell'evento *tmrAnima_Tick* una variabile *ctr* definita come *Static* e non *Dim*: in questo modo *ctr* mantiene il valore anche quando il flusso del programma esce dall'evento *Tick*. Se l'avessimo creata con una *Dim* di fatto non avrebbe mai superato il valore 1 e l'animazione non avrebbe mai funzionato. L'incremento della variabile statica avviene con *ctr=+1*.

Per fare in modo che le immagini effettivamente cambino ogni 150 millesimi di secondo non dovete fare altro che uguagliare *picPackMan.Image* (che per default già punta alla prima delle tre immagini che creano l'animazione) con i fotogrammi contenuti nella ImageList. Inoltre per direzionare correttamente Packman dovete testare la variabile *flgPM*, che puó contenere 4 valori necessari per puntare all'ImageList corretta: 1->sinistra, 2->destra, 3->in basso e 4->in alto.

Prima dell'*If...ElseIf* ricordatevi di fare il dispose delle immagini per evitare di sovraccaricare eccessivamente la memoria del Pocket PC. La *picPackMan.Image.Dispose()* é, dunque, fondamentale per evitare errori bloccanti in fase di esecuzione.

Infine se la variabile ctr ha raggiunto il suo massimo valore, cioè 4, uguagliatela di nuovo a zero per ricominciare il ciclo.

```
    Static ctr As Integer
    Dim totIL1 As Integer =
                        imaAnimaSx.Images.Count
    ctr += 1
    picPackMan.Image.Dispose()
    If flgPM = 1 Then 'sx
```

```
        picPackMan.Image =
                    imaAnimaSx.Images.Item(ctr - 1)
    ElseIf flgPM = 2 Then 'dx
        picPackMan.Image =
                    imaAnimaDx.Images.Item(ctr - 1)
    ElseIf flgPM = 3 Then 'giu
        picPackMan.Image =
                    imaAnimaGiu.Images.Item(ctr - 1)
    ElseIf flgPM = 4 Then 'su
        picPackMan.Image =
                    imaAnimaSu.Images.Item(ctr - 1)
    End If
    If ctr = totIL1 Then ctr = 0
```

SPOSTARE PACKMAN

A questo punto non ci resta che vedere come spostare Packman sul campo da gioco. Le opzioni che si possono fornire sono duplici: o via software o via hardware.

Partiamo dai pulsanti software che avete già visto in precedenza: *btnSx*, *btnSu*, *btnDx* e *btnGiu*.

Per spostare Packman sullo schermo è sufficiente modificare le coordinate della *PictureBox picPackMan* ogni volta che viene fatto clic su uno dei pulsanti. Ad esempio potete definire all'interno dell'evento click del pulsante *btnSx* che la proprietà *Left* (corrispondente all'ordinata) viene decrementata ogni volta di 20 pixel, al fine di spostare verso sinistra Packman. Ovviamente dovete preventivamente testare che non sia avvenuto lo "sfondamento" del bordo sinistro del campo da gioco con la *If picPackMan.Left - 20 >= 0 Then*. In questo evento viene ogni volta inizializzato il flag *flgPM* che permette all'evento *tmrAnima_Tick* di decidere quale "volto" dare a Packman.

```
  Private Sub btnSx_Click(...) Handles btnSx.Click
    flgPM = 1
    If picPackMan.Left - 20 >= 0 Then
      picPackMan.Left = picPackMan.Left - 20
    End If
  End Sub
```

L'algoritmo precedente deve essere opportunamente adattato negli eventi *Click* degli altri pulsanti, tenendo presente la verifica dei limiti del campo da gioco e l'incremento o il decremento delle proprietà *Left* e *Top*.

Detto questo è più efficiente, tuttavia, utilizzare un unico gestore degli eventi *Click* dei quattro pulsanti. Dovete dichiarare una procedura *btnMuovi_Click* che riporta tra parentesi gli stessi parametri degli eventi *Click* dei pulsanti cioè *ByVal sender As System.Object*, ByVal e As System.EventArgs; dopo la clausola Handles

UN NUOVO LIVELLO

Se in tmrMostro_Tick esistono le condizioni per passare ad un nuovo livello viene richiamata la procedura pNuovoLivello, che effettua le inizializzazioni necessarie per procedere.

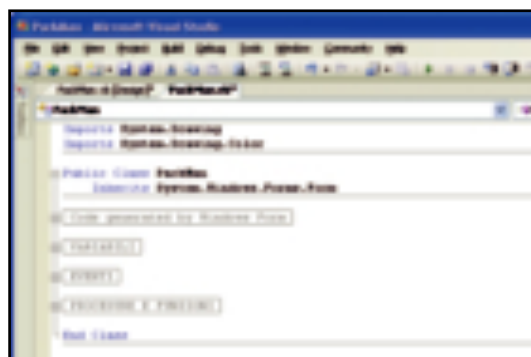



Fig. 7: Le Region sono particolarmente utili per mantenere ordinato e, quindi, più manutenibile il codice.

specificate *btnSx.Click*, *btnDx.Click*, *btnGiu.Click*, *btnSu.Click*. In questo modo si dichiara di voler utilizzare un unico gestore degli eventi e, dunque, non dovranno essere presenti i singoli eventi *Click*.
Come prima istruzione dovete, poi, specificare *Dim btnSelezione As Button = CType(sender, Button)* che serve a definire un pulsante virtuale che corrisponde esattamente a quelli creati sulla form. Facendo riferimento ad esso si possono eseguire tutte le azioni che avremmo eseguito sui singoli pulsanti. Nel codice seguente, ad esempio, potete notare che per testare qual è il pulsante selezionato in un dato momento viene testata la proprietà Text.

```
Private Sub btnMuovi_Click(ByVal sender As
        System.Object, ...) _
    Handles btnSx.Click, btnDx.Click,
        btnGiu.Click, btnSu.Click
  Dim btnSelezione As Button = CType(sender,
        Button)
  If btnSelezione.Text = "SX" Then
    flgPM = 1
    If picPackMan.Left - 10 >= 0 Then
      picPackMan.Left = picPackMan.Left - 20
  ElseIf btnSelezione.Text = "DX" Then
    flgPM = 2
    If picPackMan.Left + 10 <= 202 Then
      picPackMan.Left = picPackMan.Left + 20
  ElseIf btnSelezione.Text = "GIU'" Then
    flgPM = 3
    If picPackMan.Top + 10 <= 185 Then
      picPackMan.Top = picPackMan.Top + 20
  ElseIf btnSelezione.Text = "SU" Then
    flgPM = 4
    If picPackMan.Top - 10 >= 4 Then
      picPackMan.Top = picPackMan.Top - 20
  End If
  pMangia()
End Sub
```

Prima di passare all'esame dei pulsanti hardware considieriamo la procedura *pMangia()* che consente a Packman di mangiare le pillole colorate. Il modo più semplice di fare ciò e quello di colorare di bianco i singoli cerchi mano a mano che Packman si muove. Il codice, che qui non riportiamo, è del tutto simile a *pMettiPallini()*.

I PULSANTI HARDWARE

I Pocket PC sono solitamente dotati di alcuni pulsanti hardware; sicuramente le quattro frecce sono sempre presenti. Bene con VB.NET potete programmarle per fare spostare Packman. Per fare questo dovete programmare l'evento *KeyDown* della form, testando ogni volta qual è il tasto hardware premuto. All'interno di ciascun If dovete riportare le istruzione per il movimento di Packman. Infine dovete fare una chiamata a *pMangia()*.

```
Private Sub PackMan_KeyDown(...) Handles
        MyBase.KeyDown
  If (e.KeyCode = Keys.Left) Then
    flgPM = 1
    If picPackMan.Left - 20 >= 0 Then
      picPackMan.Left = picPackMan.Left - 20
    End If
  ElseIf (e.KeyCode = Keys.Right) Then
    ...
  ElseIf (e.KeyCode = Keys.Down) Then
    ...
  ElseIf (e.KeyCode = Keys.Up) Then
    ...
  End If
  pMangia()
End Sub
```

MUOVERE I FANTASMINI

Un videogioco senza ostacoli non è un vero videogioco. Ecco perché è importante la presenza dei fantasmini, che se "acchiappano" Packman registrano una penalità nel punteggio rosso. Le regole che abbiamo inventato sono le seguenti:

- esistono quattro livelli successivi, dove i fantasmi aumentano sempre di uno fino ad arrivare a quattro nell'ultima barriera.
- Se Packman viene mangiato quattro volte (20.000 punti rossi) il gioco finisce.
- Ad ogni cambio di livello il punteggio verde s'incrementa sempre, mentre quello di destra si azzera ogni volta.

Inserite nel progetto un controllo *Timer*, caratte-

UTILIZZARE IL SOFTWARE ALLEGATO

Come prima cosa copiate l'intero progetto Packman dal CD al vostro PC. Per avviarlo all'interno di Visual Basic.NET entrate nella cartella Packman e fate doppio clic sul file Packman.sln.

rizzato dalle seguenti proprietá:

- *(Name)= tmrMostro*
- *Enabled=True* (in questo modo il *Timer* risulta attivo in fase di esecuzione)
- *Interval=800* (frequenza in millisecondi di attivazione dell'evento *Tick*)

Ciascun fantasmino viene fatto muovere casualmente all'interno del campo da gioco grazie alla procedura *pMuoviMostro* che riceve come parametro di input la *PictureBox* corrente *picMostro* e che viene richiamata nell'evento *Tick* di *tmrMostro* tante volte quante necessarie dal livello corrente.

```
Private Sub pMuoviMostro(ByVal picMostro As
                                        PictureBox)
    picMostro.Visible = True
    Randomize()
    picMostro.Top = Int((192 * Rnd()) + 0)
    picMostro.Left = Int((208 * Rnd()) + 0)
    If (picPackMan.Top <= picMostro.Top + 25 And
      picPackMan.Top >= picMostro.Top - 25) And _
        (picPackMan.Left <= picMostro.Left + 25
 And picPackMan.Left >= picMostro.Left - 25) Then
        lblPunti2.Text += 5000
        pSuona("Ghost")
    End If
End Sub
```

Successivamente vengono calcolate le coordinate per posizionare il fantasmino in una posizione casuale del campo da gioco. A questo proposito vengono utilizzate le due funzioni *Randomize()*, per inizializzare il generatore di numeri casuali, e *Rnd()* che individua un numero a caso tra zero e 192, nel primo caso, e un altro tra zero e 208, nel secondo caso.
Infine dovete accertare se le coordinate correnti del fantasmino coincidono in parte o completamente con quelle di Packman. Se la *If* è verificata potete aggiungere 5.000 punti come penalità.
Il cambio di livello e la fine della partita, vengono controllati sempre nell'evento *Tick* di *tmr Mostro*.

```
    If lblPunti2.Text = 20000 Then
        pulsantiHW = False
        pFine(Chr(13) & "GIOCO FINITO " & _
            Chr(13) & Chr(13) & "Raggiunto " &
             Chr(13) & "il numero massimo delle" & _
            Chr(13) & " penalitá -> 20000!")
    ElseIf lblPunti1.Text = 16500 And livello = 1
                                              Then
        pNuovoLivello("SECONDO", 2)
    ElseIf lblPunti1.Text = 33000 And livello = 2
                                              Then
```

```
        pNuovoLivello("TERZO", 3)
    ElseIf lblPunti1.Text = 49500 And livello = 3
                                              Then
        pNuovoLivello("QUARTO", 4)
    ElseIf lblPunti1.Text = 66000 And livello = 4
                                              Then
        pFine(Chr(13) & "GIOCO FINITO " & _
            Chr(13) & Chr(13) & "Complimenti " &
                  Chr(13) & "hai totalizzato il" & _
            Chr(13) & " M A S S I M O !!!")
    End If
```

ANIMARE I FANTASMINI

Mentre il movimento di ciascuno fantasmino avviene grazie alla sub *pMuoviMostro*, è la *pAnimaMostro* che si cura di visualizzarne l'animazione.

```
Private Sub pAnimaMostro(...)
    Dim totMostri As Integer =
                        imaMostro.Images.Count
    ctrM += 1
    picMostro.Image =
                  imaMostro.Images.Item(ctrM - 1)
    If ctrM = totMostri Then ctrM = 0
End Sub
```

Al solito le varie immagini sono presenti in apposite ImageList. L'alternanza tra un'immagine e l'altra viene "comandata" nell'evento Tick di tmrAnima dove, a seconda del livello, viene richiamata una o piú volte la pAnimaMostro, passandole i seguenti parametri:

- la *PictureBox* del fantasmino che si vuole animare.
- La *ImageList* che contiene i fantasmini da visualizzare in successione.
- Un contatore che tiene conto dell'immagine corrente.

EVENTI SONORI

Al fine di rendere più piacevole il gioco sono stati introdotti alcuni suoni: Start.wav che segnala l'inizio di ogni partita, Mangia.wav che evidenzia la mangiata di ogni singola pillola da parte di Packman ed, infine, Ghost.wav che viene eseguito quando un fantasmino ha divorato Packman.
Per eseguire i file wav dovete utilizzare una delle funzioni API che Windows Mobile ci mette a disposizione.
La funzione *PlaySound* deve essere dichiarata in un apposito modulo di classe. Scegliete, dunque, dal menu *File-Add New Item*, fate clic su *Class* e

nella finestra che appare date un nome alla classe nel campo *Name*, ad esempio *eseguiWav*. Successivamente fate clic sul pulsante *Open*.
La *PlaySound* è una funzione nativa del sistema operativo (una cosiddetta API, Application Program Interface) ed è presente in un file esterno che deve essere referenziato. Per fare ciò dovete prima aggiungere un'istruzione di *Import* anteponendola alla dichiarazione della classe e, poi, un riferimento (*Declare*) alla DLL che contiene la *PlaySound* nel Pocket PC, ossia *Coredll.dll*.

```
Imports System.Runtime.InteropServices
Public Class eseguiWav
    Declare Function PlaySound Lib "coredll.dll" (...)
                                            As Boolean
End Class
```

Di seguito viene riportata la funzione *pSuona* utile ad eseguire i file wav che avrete preventivamente copiato nella cartella \My Documents \My Music del Pocket PC.

```
Private Sub pSuona(ByVal pWave As String)
    Dim Suono As String = "\My Documents\My
                         Music\" & pWave & ".wav"
    Dim mioWav = eseguiWav.PlaySound(Suono,
                                   IntPtr.Zero, 1)
End Sub
```

Basterà passare il nome del file sonoro (senza estensione .wav) alla *pSuona* per ottenerne l'esecuzione immediata.

IL PUNTEGGIO

Mentre le penalità sono piuttosto semplici da assegnare, perché si tratta semplicemente di verificare una coincidenza di coordinate e, poi, d'incrementare il punteggio rosso, i punti verdi richiedono maggiore impegno per la loro identificazione. Questo perché non potete semplicisticamente aggiungere un certo punteggio ad ogni spostamento di Packman, ma solo quando questi ha effettivamente mangiato una pillola. Vanno dunque esclusi gli spostamenti su spazi ormai bianchi. Come fare? Il modo più semplice sembra essere quello di utilizzare una matrice a due dimensioni, la *arRigaCol*, che dichiarate nella *Region* delle variabili come *Dim arRigaCol(183, 201) As String*.
La *arRigaCol* inzialmente contiene solo spazi vuoti, ma nella funzione *fPunti* imposta la casella corrispondente al pallino mangiato con una X, se la casella stessa è vuota, se cioè in precedenza il pallino non era già stato mangiato.

```
Private Function fPunti() As Boolean
    Dim flgEsci As Boolean
    If arRigaCol(picPackMan.Top - 1,
                    picPackMan.Left - 1) = "" Then
        arRigaCol(picPackMan.Top - 1,
                        picPackMan.Left - 1) = "X"
        lblPunti1.Text += 150
        flgEsci = True
    End If
    fPunti = flgEsci
End Function
```

Ad ogni spostamento la *If* verifica se la casella corrispondente della matrice è ancora vuota, in questo caso oltre ad impostare la X, incrementa il punteggio e imposta il *flgEsci* a *True* per indicare alla sub chiamante *pMangia*, che devono essere fatte certe operazioni.
La *fPunti* viene richiamata dalla procedura *pMangia*, a sua volta invocata ad ogni movimento di Packman, che è stata opportunamente modificata rispetto a quella della puntata precedente.

```
Private Sub pMangia()
    If fPunti() Then
        Dim i As Integer
        Dim myPen As New Pen(White)
        pSuona("Mangia")
        For i = 0 To 20
            Me.Grafici.DrawEllipse(myPen,
                    (picPackMan.Left + 6) + i / 2, _
                (picPackMan.Top + 6) + i / 2, 20 - i,
                                             20 - i)
        Next
        myPen.Dispose()
    End If
End Sub
```

E' stato introdotto un test sulla funzione *fPunti* che rilascia *True* se la casella corrispondente allo spostamento appena effettuato da Packman è vuota. In questo caso viene eseguito il file audio Mangia.wav e, poi, si procede alla cancellazione del pallino. Se *fPunti* rilascia *False* non viene fatto nulla.

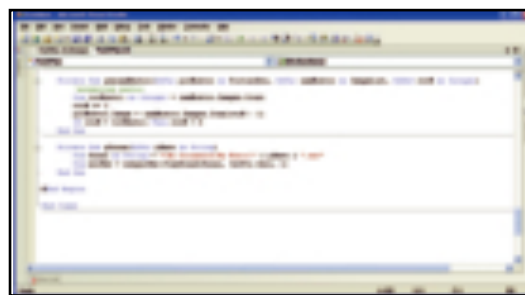



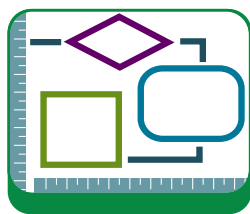
***Fig. 8:** La partita è finita, Packman è stato "mangiato" più di quattro volte!*

NOTA

DISTRIBUIRE L'APPLICAZIONE

Per distribuire l'applicazione, una volta ultimata, copiate il file PackMan.exe, presente in PackMan\bin\Release, all'interno del PPC. Inoltre dovete anche copiare nel percorso del Pocket PC \My Documents\My Music i file audio Ghost.wav, Mangia.wav e Start.wav. Nessun altro file sarà necessario perché Windows Mobile contiene già tutte le librerie indispensabili al funzionamento delle applicazioni .NET.

IL PATTERN VALUE OBJECT

GLI ERRORI DI "ALIASING" SONO TRA LE CAUSE PIÙ COMUNI DI BUG INSIDIOSI NELLA PROGRAMMAZIONE A OGGETTI. UN PATTERN SEMPLICE MA IMPORTANTE DI NOME VALUE OBJECT CI AIUTA AD ELIMINARLI DEFINITIVAMENTE.

Immagina un bravo programmatore, che dorme tranquillo dopo una giornata faticosa. Il nuovo sistema di gestione dei conti correnti è andato in produzione il giorno prima. Trattandosi di un sistema critico, l'azienda ha deciso di sperimentarlo in una sola filiale della banca che l'ha commissionato.
Durante la notte, il direttore della filiale butta giù dal letto il capoprogetto. Pare che i soldi dei nuovi conti correnti stiano misteriosamente svanendo nel nulla. A volte succede anche il contrario, e alcuni conti si ritrovano con più soldi del dovuto. Il problema è talmente spettacolare che sarebbe quasi divertente, se non fosse che le banche (poco abituate a tirare fuori soldi di tasca propria) sono completamente prive di senso dell'umorismo.
Così il capo chiama il bravo programmatore e lo scongiura di correre in ufficio prima del canto del gallo e risolvere il problema.

MONEY, MONEY MONEY

Ancora in pigiama, il nostro eroe osserva il codice del sistema. È scritto in Ruby, ma vi spiegheremo quanto basta per capirlo anche se non conoscete questo linguaggio.
La classe fondamentale del sistema è, come ci si potrebbe aspettare, quella che rappresenta i soldi. Ecco le prime righe della classe:

```
class Money
  attr_reader :cents

  def initialize(euros, cents = 0)
    @cents = euros * 100 + cents
  end

  ...
```

Cerchiamo di capire questa classe un metodo alla volta. Non è necessario che capitate ogni dettaglio, a meno che non vogliate diventare programmatori Ruby. L'importante è che vi facciate una buona idea di come funziona *Money*. *Money* rappresenta internamente il denaro come un numero intero di centesimi, conservato in un campo *@cents* (il prefisso @ in Ruby indica i campi di un oggetto). Visto che Ruby è un linguaggio dinamico, non c'è bisogno di dichiarare il tipo di *@cents*: basta assegnarli un valore.
Il valore iniziale di *@cents* viene assegnato nel metodo *initialize()*, che è l'equivalente Ruby di un costruttore. *initialize()* prende il numero di Euro e quello di centesimi, e converte tutto in un singolo numero che rappresenta i centesimi. Il numero di centesimi è opzionale: se *initialize()* viene chiamato con un solo argomento, i centesimi assumono il valore zero per default.

QUESTIONE DI CLASSE

La prima riga della classe *Money* definisce un attributo in sola lettura di nome *cents*. Per le convenzioni di Ruby, questo attributo restituisce il valore del campo con lo stesso nome, *@cents*. Ecco un semplice esempio:

```
price = Money.new(1, 30)
puts price.cents
```

Queste due righe di codice creano un oggetto di classe *Money* inizializzandolo con 1 Euro e 30 centesimi. In Ruby gli oggetti si creano attraverso il metodo *new()*, che a sua volta chiama *initialize()*. Il tutto viene convertito internamente in 130 centesimi, e *puts* (equivalente Ruby di *print*) stampa questo valore sullo schermo.
Vediamo il resto della classe Money:

```
def class Money

  ...
```

```
  def add(other_money)
    @cents += other_money.cents
  end

  def sub(other_money)
    @cents -= other_money.cents
  end

  def to_s
    return "#{@cents / 100},#{@cents % 100}€"
  end
```

I metodi *add()* e *sub()* prendono un altro *Money* e rispettivamente aggiungono o sottraggono i centesimi del secondo oggetto a quelli dell'oggetto ricevente. Il metodo *to_s()* restituisce invece una stringa che rappresenta l'oggetto. Questo metodo viene chiamato ad esempio quando si stampa un oggetto sullo schermo con *puts*. In questo caso il metodo restituisce una stringa che mostra il numero di centesimi diviso per cento (gli euro) e il numero di centesimi modulo cento (i centesimi). Vi risparmiamo i dettagli sulla formattazione di stringhe in Ruby per passare subito ad un piccolo test:

```
m = Money.new(2, 50)
puts m
```

Il risultato è:

```
2,50€
```

La classe *Money* è tutta qui, e non sembra che contenga alcun bug. Passiamo alla classe *Account*, che rappresenta un conto corrente:

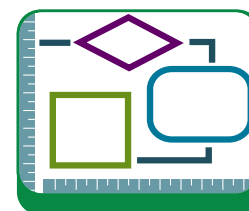
```
class Account
  attr_accessor :money

  def initialize(owner, money)

    @owner = owner
    @money = money
  end

  def to_s
    "#{@owner} possiede #{@money}"
  end
end
```

Account ha due campi: un proprietario (*@owner*) e un importo di denaro (*@money*), che è anche accessibile attraverso l'attributo *money*. Possiamo creare un account passandogli una stringa per identificare il proprietario, e un importo iniziale in denaro:

```
account = Account.new("Gino", Money.new(100,
                                        25))
puts account
```

E il risultato è:

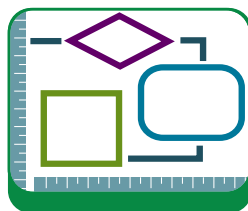
```
Gino possiede 100,25€
```

Sembra che vada tutto bene. Come è possibile che due classi così semplici nascondano dei bug? Eppure il bug esiste, e non ci resta che cercarlo nel codice che gestisce l'apertura dei conti correnti.

IL MISTERO DEI SOLDI FANTASMA

Il codice che apre un conto corrente ha una logica simile a questa:

```
INITIAL_DEPOSIT = Money.new(100)
account1 = Account.new("Franco",
                       INITIAL_DEPOSIT)
account2 = Account.new("Piero",
```

VALUE OBJECTS IN JAVA E C#

A differenza di Ruby, dove ogni variabile è un riferimento ad un oggetto, Java e C# hanno il concetto di "variabile primitiva". Le variabili primitive non sono oggetti, ma tipi semplici come int o float. Queste variabili sono automaticamente "valori". Ad esempio, non potete cambiare il numero 1: potete solo usarlo in espressioni aritmetiche per ottenere altri numeri. In linguaggi come questi, il problema dell'aliasing sorge solo quando si usano riferimenti ad oggetti.
Un Value Object deve essere immutabile, quindi è bene che i suoi campi siano delle costanti. Ruby definisce i riferimenti costanti con una convenzione (la prima lettera del loro nome è maiuscola). Altri linguaggi richiedono invece una parola chiave: Java usa final e C# usa const. Se dichiarate un riferimento final o const, non potete più cambiare l'oggetto a cui punta quel riferimento. Ma attenzione: come in Ruby, il fatto che un riferimento sia costante non significa che l'oggetto a cui fa riferimento sia costante. Per assicurarvi che un oggetto sia immutabile, verificate che i suoi campi siano variabili primitive o riferimenti costanti ad oggetti immutabili. Notate anche che molti dei tipi più usati in questi linguaggi, come String o Date in Java, sono già immutabili – quindi potete tranquillamente usarli come campi dei vostri Value Objects.
I Value Object hanno di solito un operatore di confronto che confronta i valori di due oggetti anziché i loro riferimenti. Java non ha l'overloading degli operatori, quindi non è possibile ridefinire l'operatore di confronto (==). In Java, questo operatore verifica sempre e solo l'identità, non l'uguaglianza. Per sopperire a questo problema, tutti gli oggetti Java hanno un metodo equals() che può essere ridefinito e usato al posto dell'operatore di confronto. I programmatori Java imparano ben presto (di solito a proprie spese) ad usare equals() al posto di == per confrontare tutti gli oggetti, stringhe comprese. In questo caso, C# è più flessibile e permette di ridefinire l'operatore ==.

```
INITIAL_DEPOSIT)
```

Per aprire un conto, si deve versare un deposito fisso di 100 Euro, che va consegnato in contanti al funzionario di banca. Quindi il sistema definisce una costante che contiene cento Euro (in Ruby, un riferimento il cui nome inizia con una lettera maiuscola è automaticamente una costante).
A questo punto possiamo scrivere un semplice test per verificare cosa succede se Franco, il proprietario del primo conto, decide di prelevare venti Euro:

```
account1.money.sub(Money.new(20))
puts account1
puts account2
```

Il risultato è:

```
Franco possiede 80,0€
Piero possiede 80,0€
```

Orrore! Sebbene Piero non abbia mai prelevato soldi dal proprio conto, venti dei suoi Euro sono misteriosamente svaniti nel nulla. A quanto pare, ogni volta che uno dei due correntisti effettua un'operazione sul proprio conto, l'operazione viene ripetuta identica anche sul conto dell'altro. Come è possibile?

PERCHÉ LA CLASSE MONEY?

Qualche lettore potrebbe pensare che ci stiamo complicando inutilmente la vita. Perché scrivere una classe Money? Non si potrebbe semplicemente usare un banale numero in virgola mobile per indicare gli Euro? I numeri sono già dei Value Objects - anzi, in molti linguaggi sono semplicemente delle variabili primitive, e non essendo oggetti non hanno problemi di aliasing. Quindi un numero sembrerebbe un candidato ideale per rappresentare un importo monetario.
Uno dei motivi per scrivere una classe Money è che questa classe conterrà probabilmente anche della logica complessa, come quella per le conversioni di valuta. Ma anche se non avete questo problema, vi conviene comunque stare alla larga dai numeri in virgola mobile. Ad esempio, guardate questo programmino che somma 0,99 Euro per cento volte di seguito:

```
money = 0
100.times do
  money += 0.99
end
puts money
```

Scrivere 100.times è un modo (piuttosto originale) per ripetere un'operazione per cento volte in Ruby. L'istruzione puts stampa il risultato sullo schermo. Quale risultato vi aspettate? Se vi sembra ovvio rispondere "99 euro", provate a far girare il programma:

```
98.9999999999999
```

Abbiamo appena perso un decimillesimo di miliardesimo di Euro per colpa dell'arrotondamento. A parte il fatto che la banca riterrebbe la perdita inaccettabile ("Qui non siamo tipo da buttare via i soldi", avrebbero detto), questo risultato inatteso scombina anche tutti i test, i report e i conteggi di verifica. Per questo motivo, conviene sempre usare numeri interi per rappresentare i soldi. La soluzione ideale sarebbe probabilmente che il linguaggio fornisse già un proprio tipo per rappresentare i soldi – ma stranamente, quasi nessun linguaggio lo possiede.

PROBLEMI DI MEMORIA

Cerchiamo di capire cosa succede in memoria quando i due *Account* vengono creati. In un linguaggio a oggetti come Ruby, non si manipolano mai direttamente gli oggetti. Invece, si manipolano delle variabili (e delle costanti) che sono dei riferimenti agli oggetti in memoria. Ad esempio, *INITIAL_DEPOSIT* è un riferimento costante ad un *Money*. Questo riferimento viene passato ai costruttori dei due Account. Vediamo ancora una volta cosa succede dentro ciascun Account:

```
class Account
  def initialize(owner, money)
    @owner = owner
    @money = money
  end

  ...
```

In Ruby, il passaggio degli argomenti e l'assegnamento avvengono sempre copiando i riferimenti. Quando chiamiamo *initialize()* attraverso *new()*, stiamo copiando il riferimento all'oggetto in un argomento di nome *money*, che subito dopo viene a sua volta copiato in un campo di nome *@money*. Quindi il riferimento viene copiato più volte, ma l'oggetto referenziato è sempre lo stesso. La Figura 1 mostra la situazione in memoria: due *Account* che contengono ciascuno un riferimento allo stesso oggetto Money, che è anche referenziato dalla costante *INITIAL_DEPOSIT*.

LE COSTANTI NON COSTANTI

Ricapitoliamo: abbiamo creato un *Money*, lo abbiamo referenziato tramite la costante *INITIAL_DEPOSIT* e lo abbiamo passato ad *Account.new()*. Da qui il *Money* viene passato ad *Account.initialize()*, che lo assegna al riferimento *@money*. Lo stesso avviene per il secondo oggetto. Quindi, una volta che abbiamo finito di costruire gli account, abbiamo tre oggetti che referenziano lo stesso *Money*.

Quando Franco decide di prelevare venti Euro, l'operazione viene delegata all'oggetto Money, che sottrae venti Euro da sé stesso. Ma visto che i soldi di Piero sono in realtà rappresentati dallo stesso oggetto che rappresenta i soldi di Franco, anche Piero scopre con costernazione che il suo conto in banca è diminuito. Peggio ancora: visto che anche il deposito iniziale è un riferimento allo stesso oggetto, anch'esso è ora diminuito di 20 Euro. Tutti i conti che verranno aperti d'ora in poi conterranno solo 80 Euro:

```
account1.money.sub(Money.new(20))
puts account1
puts account2
puts INITIAL_DEPOSIT
```

Il risultato è:

```
Franco possiede 80,0€
Piero possiede 80,0€
80,0€
```

Se non avete ancora molta dimestichezza con la programmazione a oggetti, potrebbe sembrarvi strano che il valore di *INITIAL_DEPOSIT* possa cambiare. Non avevamo detto che questo riferimento è una costante? Ancora una volta, l'equivoco nasce dalla differenza tra riferimenti e oggetti. Visto che *INITIAL_DEPOSIT* è un riferimento costante, Ruby protesterà se cerchiamo di assegnargli un oggetto diverso da quello con cui lo abbiamo inizializzato. Questo però non vuol dire che lo stato dell'oggetto *Money* referenziato non possa cambiare. Se il valore di uno dei campi del *Money* cambia, il povero riferimento *INITIAL_DEPOSIT* non ha nemmeno modo di accorgersene. In effetti, questo è proprio quello che succede quando chiamiamo il metodo *add()* dell'oggetto *Money*

IL PROBLEMA E LA SOLUZIONE

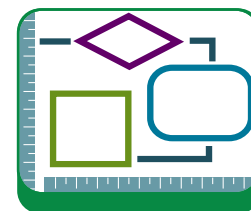
Problemi come questo sono comuni. Si chiamano problemi di "aliasing", perché derivano dal fatto che più riferimenti sono in realtà nomi diversi ("alias") per lo stesso oggetto. L'aliasing può diventare un problema se perdiamo di vista le interazioni tra gli oggetti nel nostro sistema. Un metodo all'apparenza innocuo può modificare l'oggetto che gli viene passato, causando bug come quello che ha costretto il nostro programmatore a correre in ufficio all'alba.
In molti casi, la soluzione di questo problema è semplicemente prestare attenzione agli oggetti e a chi li modifica. Ma esiste anche un pattern che può eliminare completamente il problema in caso di oggetti come *Money*.

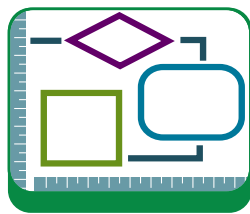
IL PATTERN VALUE OBJECT

Un *Value Object* è un oggetto che obbedisce ad un paio di semplici regole. Primo: il suo valore non cambia mai. Secondo: due Value Objects sono uguali se hanno lo stesso valore.
Partiamo dalla prima caratteristica. Il problema della classe *Money* è che oggetti diversi possono avere riferimenti allo stesso *Money*. Se uno di questi oggetti cambia il *Money*, anche gli altri oggetti lo vedranno cambiare. Questo è esattamente il problema di aliasing che abbiamo visto poco fa. Ma se il *Money* non può cambiare, allora l'aliasing non è più un problema. Gli oggetti che non possono cambiare valore si definiscono oggetti "immutabili".
Come si fa a rendere immutabile un *Money*? La classe *Money* è già quasi immutabile: non espone proprietà in scrittura, e i suoi campi (come tutti i campi in Ruby) sono privati. L'unico problema sono i metodi *add()* e *sub()*, che modificano il valore dell'oggetto. Possiamo riscrivere questi due metodi perché restituiscano dei nuovi *Money* anziché modificare quello esistente:

```
class Money
  attr_reader :cents
  def initialize(euros, cents = 0)
    @cents = euros * 100 + cents
  end
  def add(other_money)
    Money.new(0, @cents + other_money.cents)
  end
  def sub(other_money)
    Money.new(0, @cents - other_money.cents)
  end
  def to_s
    return "#{@cents / 100},#{@cents % 100}€"
  end
end
```

Abbiamo omesso la parola chiave *return*, che in Ruby è opzionale. Ora *add()* e *sub()* restituiscono un nuovo *Money*, quindi non è più possibile modificare un *Money* dopo averlo creato. Questo significa che dobbiamo anche modificare la classe *Account* per incapsulare completamente il *Money*:

```
class Account
  def initialize(owner, money)
    @owner = owner
    @money = money
```

IL LINGUAGGIO RUBY

**Se volete provare gli esempi di questo articolo sul vostro computer, potete scaricare il linguaggio Ruby dal sito http://www.ruby-lang.org.
In particolare, se usate Windows vi consigliamo il One-click Ruby Installer. Quando avete installato il linguaggio, potete lanciare un programma con il comando:
ruby nome_del_programma.rb**

```
  end

  def add(money)
    @money = @money.add(money)

  end

  def sub(money)
    @money = @money.sub(money)
  end

  def to_s
    "#{@owner} possiede #{@money}"
  end
end
```

Ci siamo quasi. Resta solo da rispondere ad un'ultima domanda: quando possiamo dire che due *Money* sono uguali?

UGUALI E IDENTICI

Proviamo a confrontare due oggetti Money che rappresentano entrambi la somma di 10,15 Euro. Usiamo l'operatore di confronto, che in Ruby è rappresentato da un doppio "uguale":

```
m1 = Money.new(10, 15)
m2 = Money.new(10, 15)
puts m1 == m2
```

L'operatore di confronto non confronta il valore degli oggetti, ma il valore dei riferimenti *m1* e *m2*. Dato che questi riferimenti puntano a due oggetti *Money* diversi in memoria, questo codice stampa *false* anche se i due oggetti hanno valori uguali in tutti i campi. Con tutta probabilità, questo non è quello che gli utenti della classe *Money* si aspettano. I soldi sono soldi, e un importo di 10,15 Euro non è diverso dall'altro.

Quando abbiamo a che fare con dei valori, ci aspettiamo di poter ignorare il concetto di identità. Se due Value Object hanno lo stesso valore, non ci interessa che siano oggetti diversi in memoria. In Ruby possiamo risolvere il problema grazie all'overload degli operatori. Basta aggiungere poche righe di codice a *Money* per ridefinire l'operatore di confronto:

```
class Money
  def ==(other)
    return false if other.class != Money
    return cents == other.cents
  end

  ...
```

PARTICOLARITÀ DEL LINGUAGGIO RUBY

L'operatore di confronto in Ruby è in realtà un metodo che viene chiamato sul primo oggetto, passandogli come argomento il secondo oggetto. L'implementazione di default di ==() restituisce *true* solo se i due oggetti sono *identici*, cioè se sono effettivamente lo stesso oggetto in memoria. Noi vogliamo invece restituire *true* anche se i due oggetti sono *uguali*, cioè se hanno lo stesso valore. La prima riga del metodo può sembrarvi strana se non conoscete Ruby. In questo linguaggio si può mettere l'*if* dopo la condizione, quindi questa riga si legge: "Restituisci *false* se l'altro oggetto non è di classe *Money*". La seconda riga restituisce true o false a seconda che i due importi di denaro siano uguali, cioè contengano lo stesso numero di centesimi.
Ora il confronto tra due *Money* uguali ma non necessariamente identici restituisce *true*. Il nostro bravo programmatore merita di tornare finalmente a casa e dormire per il resto della giornata.

CONCLUSIONI

Ricapitoliamo: un Value Object è un oggetto la cui uguaglianza dipende dal valore piuttosto che dall'identità. I Value Object sono tipicamente immutabili, quindi non hanno problemi di aliasing. Di solito sono anche "piccoli", nel senso che rappresentano quantità semplici come un importo di denaro, o una stringa, o una data. Niente vi impedisce di scrivere Value Objects con dozzine di campi, ma in questo caso state attenti alle prestazioni: ricordate che ogni piccola variazione ad un Value Object vi costringe a creare un nuovo Value Object, e creare migliaia di oggetti molto grossi può essere dispendioso.
Ciononostante, quando scrivete una nuova classe riflettete sempre sull'opportunità di implementarla come un Value Object e risolvere alla radice il problema dell'aliasing.
Come tutti i pattern, il Value Object rappresenta una soluzione ben rodata ad un problema comune. Nonostante che in questo articolo abbiamo usato RUBY per descrivere il problema e risolverlo tramite un pattern, in realtà è importante sottolineare l'importanza dell'algoritmo rispetto al linguaggio. Con poche modifiche il pattern può essere facilmente applicato ad ogni linguaggio

Paolo Perrotta

SOFTWARE SUL CD

Spices.NET Suite 5.2.0.8

4 PROGRAMMI IN UNO

La suite raccoglie un obfuscator, un decompiler, un modeler e un investigator.L'obfuscator consente di "criptare" il binario di un codice .NET di modo che non sia facilmente possibile disassemblarlo ed ottenerne il codice originale. Il decompiler viceversa consente di decompilare un software .NET e visionarne la struttura. A completare la suite due strumenti utilissimi, il primo: il modeler consente di progettare il proprio applicativo ed il secondo di navigarne i metadati. Si tratta una suite veramente ben studiata che fornisce al programmatore una base certa quando si vogliono integrare o modificare programmi scritti da altri

Directory:Rspices.zip

XAMPP 1.5.5

IL TUO SERVER PERSONALE!

Installando questo software avremo a disposizione un completo ambiente per lo sviluppo di applicazioni web basate su PHP e MySQL, ossia una delle piattaforme più usate del web. I programmi installati saranno: Apache 2.2.3 (server web), MySQL 5.0.27 (motore SQL), PHP 5.20/4.4.4 (linguaggio di scripting), phpMyAdmin 2.9.1.1 (tool web per amministrare MySQL), FileZilla FTP Server 0.9.20 (server FTP), OpenSSL 0.9.8.d (librerie per l'implementazione del protocollo SSL). La presenza di un pannello di controllo, inoltre, semplificherà l'avvio e l'arresto dei vari server.

Directory: xampp-win32-1.5.5-installer.exe

PIVOT 1.4.0.4

IL BLOG CHE AVANZA

I blog hanno rappresentato e rappresentano tuttora una delle grandi rivoluzioni della rete. La parola d'ordine per ogni blog è "semplicità". Semplicità di gestione, di installazione, e di utilizzo. Pivot è orientato in questa direzione ma soprattutto non necessita di nessun database di supporto, per cui si presta ad essere utilizzato in sistemi di hosting a basso costo che non offrono db nei loro pacchetti base

Directory: pivot_1404_full.zip

DASHCOMMERCE 2.0.1

IL SISTEMA DI ECOMMERCE IN ASP.NET

Dashcommerce è un ottimo sistema di ecommerce scritto completamente in .NET e con l'enorme vantaggio di essere completamente Free. Il sistema è interessante e le caratteristiche esposte ne consentono l'uso anche in fase di produzione. La disponibilità del codice lo rende idoneo ad essere adattato per moltissime esigenze

Directory:dashCommerce_2.0.1.zip

DOJO TOOLKIT 0.9.0

IL FRAMEWORK PER JAVASCRIPT

Javascript dopo un periodo di appannamento sembra essere tornato alla ribalta come motore dello sviluppo di Ajax. Questa rinnovata popolarità ha dato il via alla nascita di una serie di framework che facilitano la programmazione javascript. Fra i tanti spicca dojo. Prima di tutto per la sua innata propensione alla programmazione ad eventi, ma anche e soprattutto perché consente in una qualche misura di utilizzare la tecnologia Ajax in push invece che in pop. Ovvero in modo tale che sia il server ad inviare i dati al client e non viceversa. In realtà questa possibilità è in via di sviluppo e appare soltanto come una delle tante tecnologie che Dojo offre ma se affiancata a tutto il resto, proprio questa caratteristica non è da sottovalutare

Directory: dojo-release-0.9.0.tar.gz

DOTLUCENE 2.0

IL SEGUGIO TROVA TUTTO

Una delle librerie che ha attratto l'attenzione dei programmatori di tutto il mondo nell'ultimo anno è Lucene. Si tratta di una libreria che consente di indicizzare in modo ottimale il contenuto di un Hard Disk per poi poter effettuare delle ricerche molto veloci sull'indice. Lucene è stata utilizzata in progetti di grandi dimensioni come ad esempio Beagle. Ne esiste anche un porting in ambiente .NET che è appunto quello che vi presentiamo. Utilissima in tutti quei casi in cui si vogliano dotare le proprie applicazioni di funzioni di ricerca "libera" sui contenuti di un supporto o su

una parte di esso.
Directory:Lucene.Net-2.0-002-27Nov06.src.zip

SHARP DEVELOP 2.2.1

L'ALTERNATIVA A VISUAL STUDIO

Se siete programmatori C# o VB.NET ma non volete utilizzare gli ambienti di Microsoft perché troppo complessi per le vostre esigenze, SharpDevelop si rivela un'ottima alternativa. Si tratta di un ambiente leggero, visuale, rad, dotato di tutte o quasi tutte le caratteristiche di ambienti molto più costosi. Il software è completamente opensource. Se credete che questo possa essere sinonimo di scarsa affidabilità sappiate che state commettendo un grave errore. SharpDevelop è decisamente un ambiente professionale, ben studiato e progettato, inoltre il team di sviluppo è particolarmente attivo
Directory: SharpDevelop_2.2.1.2648_Setup.msi

SPRING 2.0.2

IL FRAMEWORK DEI FRAMEWORK

Spring è un Framework che riunisce sotto un unico cappello una serie di strumenti già esistenti mettendoli ingrando di comunicare in modo corretto fra loro. Inoltre Spring implementa il pattern Ioc, inversion of control che garantisce un alto grado di disaccoppiamento e una maggiore manutebilità del codice
Directory: spring-framework-2.0.2-with-dependencies

PHTML ENCODER 5.1

PER CRIPTARE LE VOSTRE PAGINE PHP

Avete necessità di proteggere i vostri script PHP da occhi indiscreti? Ecco a voi un encoder che vi consente di codificare i vostri script prima di distribuirli. Poiché La tecnica usata è relativa proprio a PHP, i vostri script "offuscati" funzioneranno su qualunque sistema sia Windows che Linux
Directory: phtmlenc51.zip

NUNIT 2.4.3

IL COSTRUTTORE DI TEST

Qualche volta, anzi molto spesso vi troverete a costruire dei test per "stressare" la vostra applicazione .NET e vedere come reagisce. Nunit è un tool opensource che vi mette a disposizione un linguaggio per scrivere procedure di test. Inoltre consente di visualizzare i dati tramite un'interfaccia grafica
Directory: NUnit-2.4.3-net-2.0.zip

DIALOG BLOCK 4.17

IL RAD PER LE WXWIDGETS

Da un po' di tempo ormai su ioProgrammo ci stiamo occupando di wxWidgets. Si tratta di un framework veramente completo per la costruzione di applicazioni c++ multi piattaforma. L'orientamento è quello verso la facilitazione della creazione di interfacce grafiche. Nonostante la ricca varietà di componenti presenti nel framework mancava ancora un tool per creare le interfacce in modo RAD. A questa mancanza sopperisce Dialog Block. Un IDE per lo sviluppo di applicazioni basate su WxWidgets e basato sul modello RAD. E' sufficiente trascinare i componenti sulla form per disegnare la propria applicazione.
Directory: DialogBlocks-4.17-Unicode-Setup.exe

DIA 0.9.5

CREA DIAGRAMMI DI FLUSSO IN MODO SEMPLICE

Molto spesso nella programmazione di un software, o anche semplicemente nell'illustrare il ciclo di vita di un'applicazione o di una situazione si fa ricorso a diagrammi di flusso. Uno dei leader indiscussi del mercato in questo settore è senza dubbio Visio di Microsoft.Dia non pretende di avere le stesse funzionalità di Visio e neanche la stessa complessità ma si pone come un software leggero diffuso sotto licenza OpenSource che può essere usato in modo conveniente in tutte quelle situazioni in cui è necessario generare un diagramma di flusso velocemente
Directory: dia-0.95-1.tar.bz.2

EMULE SRC 1.0

I SORGENTI DI EMULE!

In questo numero di ioProgrammo abbiamo analizzato il comportamento di una rete PHP, seguendo passo passo il diagramma di flusso del programma più conosciuto in questo ambito: eMule. Qui vi presentiamo i sorgenti del programma, che potete analizzare per realizzare un vostro client oppure personalizzare secondo le vostre esigenze. Si tratta di un ottimo esempio di programmazione, che consente a molti di apprendere i principi base del funzionamento di una rete P2P.
Directory: eMule0.47c-Sources.zip

NHIBERNATE 2.0

COME HIBERNATE MA PER .NET

Neanche .NET sfugge alla logica secondo cui i database relazionali sono difficilmente rappresentabili come oggetti. Ed ecco che arriva NHibernate che seguendo lo stesso paradigma di Hibernate garantisce il mapping fra oggetti ed elementi di un database relazionale superando i limiti imposti dalle due diverse tecnologie. Il tool è particolarmente interessante, sia per l'elevato grado di disaccoppiamento che offre tra gli elementi del database e le classi che lo gestiscono, sia per la capacità di trasformare dati tipicamente trattati in modo relazionale in classi ed oggetti
Directory: NHibernate-1.2.0.GA.msi

PBEANS 2.0.2

PERSISTENZA DEI DATI SENZA PROBLEMI

Pbeans è una libreria scritta in Java che consente di implementare la persistenza dei dati nelle nostre applicazioni. Grazie a Pbeans, normali tabelle, colonne e campi di un database relazionale possono essere trattati come oggetti e classi, inoltre lo stato dell'applicazione può essere salvato in un database. Si tratta sicuramente di un bel vantaggio. Non ha certo velleità di sostituire Hibernate o Castor, tuttavia è un buon tool, rapido ed efficiente da usare in progetti di dimensioni tali da non giustificare l'uso di strumenti così complessi
Directory: pbeans-2.0.2.zip

FIREBIRD 2.0.0.12748-0

IL DATABASE VELOCE COME UN FULMINE

Ha attraversato una serie incredibile di disavventure. Acquistato da Borland è poi tornato ad essere OpenSource. Presente sul mercato da tempo immemorabile è riuscito a sopravvivere alle sue varie vicissitudine solo grazie alle sue grandi doti tecniche. E' un database velocissimo e ultraleggero, dotato però di tutte le funzioni di un server professional
Directory: Firebird-2.0.0.12748-0-Win32.exe

DADABIK 4.2

UN CREATORE DI INTERFACCE VERSO DATABASE

Si tratta di una web application scritta in PHP che consente di costruire altre web application basate su database. Ad esempio se volete costruire un sito che esponga

un catalogo multimediale, non vi resta che informare Dadabik di quali campi si compone il catalogo in questione, di quali funzionalità volete che il vostro sito sia dotato, e lasciare a DadaBik il compito di generare la vostra interfaccia. Non è necessario avere competenze estese di programmazione, l'uso di DadaBik è piuttosto semplice.
Directory: dadabik_4.2.zip

VISUALWX 087.6
IL RAD PER LE WXWIDGETS

Le WxWidgets sono una straordinaria libreria che consente di sviluppare applicazioni grafiche multi piattaforma in C++ con pochissimo sforzo. Unica pecca era fino a ieri l'assenza di un ambiente RAD per la costruzione delle form. Questa limitazione è adesso superata, infatti grazie VisualWX è ora possibile disegnare con pochi colpi di mouse la propria applicazione per poi riempire di "contenuti" gli eventi associati ai vari elementi presenti sulla form.Si tratta di una comodità non da poco se si pensa a quanto facile sia sviluppare per windows e linux contemporaneamente grazie proprio alle wxwidgets
Directory: VisualWx_087-60.exe

THE GIMP TOOLKIT 2.10.13
IL TOOLKIT PER LA CREAZIONE DI INTERFACCE GRAFICHE

GTK+ è un framework nato per facilitare la creazione di interfacce e basato su componenti. Tra le altre cose GTK+ è basato su GLIB che rappresenta il core dello sviluppo di Gnome. Si tratta di un framework dalle caratteristiche piuttosto complesse ma che fornisce un supporto completo per lo sviluppo di software usabile e graficamente accattivante
Directory: gtk+-2.10.13-setup.exe

WXWIDGETS 2.8.4
COMODE LIBRERIE PER LO SVILUPPO DI INTERFACCE GRAFICHE

Interessantissime queste librerie, più volte le abbiamo utilizzate all'interno di ioProgrammo per realizzare degli esempi. Si tratta di librerie che consentono la creazione di interfacce grafiche, possono essere utilizzate da C++ ma anche da altri linguaggi come ad esempio Python. La cosa estremamente interessante è che consentono lo sviluppo di applicazioni completamente multi piattaforma, sono disponibili infatti sia in ambiente unix che in ambiente Windows.
Directory: wxMSW-2.8.4-Setup.exe

GLUT 3.7
THE OPENGL UTILITY TOOLKIT

Facile, intuitivo, performante e solido. Queste tutte le caratteristiche che fanno di Glut uno dei migliori framework per la creazione di interfacce grafiche basate su OpenGL.Glut non esporta moltissime funzioni, ma si concentra sullo sviluppo delle interfacce, e consente al programmatore di apprendere le basi della programmazione OpenGL in modo rapido. E' un ottimo framework da utilizzare in tutti quei casi dove un'eccessiva complessità rappresenterebbe semplicemente un ostacolo allo sviluppo
Directory: glut37.zip

J2ME POLISH 2.0
IL COSTRUTTORE DI GUI PER DEVICE MOBILI

J2ME polish è forse un precursore dei tempi. Si tratta di un software il cui scopo è supportare il programmatore nella creazione di interfacce destinate a dispositivi portatili quali ad esempio cellulari o palmari. Ovviamente la base su cui si fonda è J2ME ormai onnipresente in qualsiasi applicazione per dispositivi del genere, tuttavia invece la definizione dell'interfaccia basa le sue caratteristiche su semplici file HTML. Interessante è il fatto che J2MEpolish sia distribuito sotto licenza GPL
Directory: j2mepolish-2.0-RC.jar

JAVA SE DEVELOPMENT KIT 6
IL COMPILATORE INDISPENSABILE PER PROGRAMMARE IN JAVA

Se avete intenzione di iniziare a programmare in Java oppure siete già dei programmatori esperti avete bisogno sicuramente del compilatore e delle librerie Java indispensabili.
Sotto il nome di Java SE Development Kit vanno appunto tutti gli strumenti e le librerie nonché le utility necessarie per programmare in JAVA. L'attuale versione è la 6.0 update 3, ovvero la nuovissima release densa di innovazioni e molto più legata al desktop di quanto non fossero tutte le precedenti
Directory: jdk-6-windows-i586

ECLIPSE SDK 3.2.2
L'IDE TUTTOFARE

Eclipse è un progetto completo portato avanti da Eclipse Foundation con la collaborazione di una miriade di aziende fra cui IBM, Adobe, Sun e che si è prefissata lo scopo di creare un IDE estendibile per plugin adattabile a qualunque tipo di linguaggio o tecnologia. Di default Eclipse si propone come IDE per Java ed è qui che da il meglio di se. Ma proprio grazie ai suoi plugin è possibile utilizzarlo come ambiente di programmazione per PHP, per C++, per Flex e per molti altri linguaggi ancora. **Directory: eclipse-SDK-3.2.1-win32.zip**

PHP 5.2.1
IL LINGUAGGIO DI SCRIPTING PIÙ AMATO DEL WEB

Sono tre le colonne portanti di Internet: PHP, APACHE e MySQL. Certo la concorrenza è forte. Asp.NET e SQL Server avanzano con celerità, ma a tutt'oggi non si può affermare che i siti sviluppati in PHP costituiscano la stragrande maggioranza di Internet. Quali sono le ragioni del successo di cotanto linguaggio? Prima di tutto la completezza. PHP ha di base tutto quello che serve ad un buon programmatore, raramente è necessario ricorrere a librerie esterne, Il secondo punto di forza del linguaggio sta nella sua capacità di poter essere utilizzato sia in modo procedurale che nella sua forma ad oggetti certamente più potente e completa. Esiste un terzo di punta di forza essenziale che è quello riguardante la curva di apprendimento. PHP è in assoluto uno dei linguaggi con la curva di apprendimento più bassa nel panorama degli strumenti di programmazione.
Directory: php-5.2.0-Win32.zip

MYSQL 5.0.27
IL PRINCIPE DEI DATABASE

Indispensabile per programmare web application in tecnologia PHP. Nonché non sia possibile utilizzare altri database, ma MySQL e PHP rappresentano veramente un binomio inscindibile. L'integrazione fra questo database e il linguaggio di scripting più usato sulla rete è talmente alta da fare divenire quasi un obbligo l'uso congiunto di questi due strumenti
Le ultime versioni inoltre supportano molto bene anche Windows
Directory: mysql-5.0.27-win32.zip

CORRELAZIONE TRA TESTI

NELL'AMBITO DELLA LINGUISTICA MATEMATICA È PARTICOLARMENTE INTERESSANTE IL PROBLEMA DEL CONFRONTO TRA TESTI E DELLA CONSEGUENTE VALUTAZIONE DI ELEMENTI COMUNI. VEDIAMO COME AFFRONTARLO INFORMATICAMENTE

La vertiginosa espansione della società dell'informazione e della comunicazione pone quotidianamente nuove sfide e nuovi problemi con i quali confortarsi. L'automatica elaborazione di testi e la comparsa di internet hanno aumentato le quantità di dati, in particolare testi da trattare. Rispetto ai testi si sono intensificate delle attività che già in passato si svolgevano e se ne sono presentate di nuove. La ricerca di informazioni in scritti cartacei in passato era un compito tanto più difficile quanto più si aveva a che fare con supporti, come registri e libri poco indicizzati e numerosi. Oggi la ricerca di informazioni all'interno dei testi digitali può contare su tecniche avanzate. Per ricerche all'interno delle nostre personali memorie ci sono specifiche routine del sistema operativo; mentre nel vasto mondo di internet si può contare sui motori di ricerca. Comunque, anche nell'era digitale cominciano ad essere sempre più numerosi i casi in cui tali attività di ricerca diventano difficoltose se non impossibili. Altri problemi connessi alla lingua sono la tanto invocata traduzione automatica che nel corso di molti anni ha registrato notevoli miglioramenti ma che ancora per alcuni casi presenta evidenti malfunzionamenti. Lo studio grammaticale di testi è alla base delle tecniche che esaminano linguaggi e traduttori, come ad esempio tutto il processo che porta dalla scrittura di un programma per computer alla sua esecuzione. Particolarmente rilevanti sono anche le analisi quantitative della lingua che sono alla base di qualsiasi approfondimento circa lo sviluppo di tecniche legate alla informazione presente in testi. Molto usate sono al proposito le tecniche di clustering che sulla base delle informazioni che si riescono ad estrarre dai testi sono in grado di raggruppare e correlare tra loro diversi testi rispetto, ad esempio, al loro autore. Insomma, mi sembra che sia chiaro quanta energia intellettuale si stia convogliando verso la semplice parola, anzi anche verso la più elementare lettera, al fine di trattare una grande quantità di problemi. Nel presente articolo dopo aver dato un riferimento generale circa la linguistica matematica analizzeremo più da vicino un problema concreto in cui si vogliono confrontare due testi, che possono essere due poesie tanto quanto due post presi a caso in un forum o su diversi forum e verificare se è possibile quantificare eventuali attinenze e relazioni tra i due.

BREVI CENNI STORICI

I primi studi concreti sulla lingua risalgono agli indiani che si occuparono di analizzare testi da un punto di vista grammaticale per meglio comprenderli e trasmetterli. Per molto tempo il solo tipo di studio è stato di questo tipo. Nel corso degli anni, prima che la linguistica si strutturasse come una scienza autonoma, gli studi si orientarono in diverse direzioni. La prima è la descrizione della lingua in determinato periodo della storia in cui si confrontava la lingua scritta con la lingua parlata, le due spesso divergevano tra loro; si introdussero così i primi studi di fonetica con una classificazione delle parole per suoni che emettevano e con tecniche che li riproducevano (come labbiale, gutturale e altri). La seconda è un'analisi storica di come si è evoluta, che presuppone anche uno studio etimologico sulle singole parole. La terza che è di tipo comparativo si avvicina maggiormente ai nostri scopi. Soltanto che storicamente la comparazione riguardava soprattutto i rapporti tra lingue e dialetti diversi, cosicché si svilupparono i primi rudimentali ma strutturati studi per la traduzione. Una sintesi degli studi fatti nel corso degli anni porta nel primo quarto del secolo 19mo secolo ad una lavoro sistematico di alcuni aspetti della lingua cosicché si possa parlare di linguistica. Anche se prevalentemente si trattava sempre di comparare lingue differenti. Si introduce anche la fonetica strutturale, ossia lo studio della lingua attraverso apparecchi, che investiga circa la produzione dei suoni e delle loro qualità acustiche che non sono mai uguali anche per parlanti della stessa lingua, addirittura per lo stesso parlante. La linguistica strutturale appare con l'importante contributo di Ferdinand De Saussure. Egli dà il via ad una nuova

epoca, introduce l'importante principio secondo cui la lingua non è un insieme di elementi bensì un sistema in evoluzione dotato di regole, e che bisogna perseguire l'analisi di intere strutture e non di singoli elementi tra loro scollegati, anche per questo gli studi di questo studioso sono conosciuti come strutturalismo. Un altro importante contributo fu dato dalla scuola di Praga con a capo Nikolaj S. Trubeckoj dalla quale emergeva come la lingua è un sistema di sistemi (fonologico, morfologico e sintattico).

FONOLOGIA, MORFOLOGIA

Suddividere un enunciato secondo diversi livelli è un problema conosciuto come segmentazione. L'unità elementare in cui suddividere è il fonema. Lo studio fatto con l'ausilio di macchine ha portato a classificare per ogni lingua i vari fonemi. La cosa interessante è che se anche una stessa persona pronuncia più volte la stessa parola lo spettro (un'analisi quantitativa) dei singoli fonemi non sarà mai uguale. I fonemi da soli non hanno significato. Lo assumono quando sono raggruppati tra loro. L'unità minima che abbia significato è il morfema. Una parola (o con nuova terminologia sintagma minimo) di norma è costituita da più morfemi.

Ciò che è più vicino al nostro mondo è la linguistica matematica, ossia gli studi linguistici con metodi matematici. Come per molte discipline anche questa è apparsa in contemporanea all'apparizione e al successivo potenziamento dell'elaboratore. I primi studi furono dei sovietici. Qui si introducevano elementi statistici nella struttura di un testo. Concetti che da sempre hanno trovato resistenze in campo umanistico ma che sono stati molto utili per alcune analisi quantitative. Sono così stati introdotti: la teoria dell'informazione, la traduzione elettronica e altri metodi. Si è parlato per la prima volta di entropia della lingua che misura lo stato di indeterminazione di ciò che nelle lingua è arbitrario.

SINTASSI

La sintassi descrive come parole sono raggruppate secondo regole grammaticali. Per esprimere uno stesso concetto esistono diverse forme sintattiche, tutte valide. L'analisi sintattica secondo uno schema tradizionale prevede la scomposizione della proposizione in parti, la sistemazione di queste ultime secondo differenti categorie e la dimostrazione di come sono tra loro collegate le parti. La sintattica strutturale tenta di fare chiarezza su alcuni aspetti, ad esempio sul numero e l'ordine gerarchico delle unità sintattiche di una lingua.

UN ESEMPIO DI ALGORITMO DI CLUSTERING

In uno studio, che mi ha visto coinvolto personalmente, è stato dimostrato che in testi, purché di significativa lunghezza, l'informazione è contenuta anche nelle frequenze delle singole lettere dell'alfabeto. In altri termini è possibile distinguere due scritti, o meglio gli autori, semplicemente analizzando un vettore delle frequenze delle lettere dell'alfabeto degli scritti. Lo studio è stato sviluppato su testi della Grecia antica e con tecniche di clustering; sono stati individuati gruppi che corrispondevano ai singoli autori esaminati. In particolare una volta prodotta l'impronta per ogni testo, ossia il vettore delle frequenze delle 24 lettere greche, sono state applicate alcune tecniche di clustering che hanno evidenziato i gruppi per autore. Tale metodo ha consentito anche l'attribuzione di un testo la cui paternità era contesa tra più autori. Tecniche geometriche conosciute come PCA (principal componenti analisys) hanno permesso con limitata perdita di informazioni di proiettare tali impronte che di fatto erano descritte in uno spazio a 24 dimensioni (numero di componenti del vettore) in uno spazio bidimensionale di facile fruizione visiva. Anche questo passaggio che come detto prevede una perdita di informazione ha mostrato il raggruppamento dei testi ridotti a miseri puntini in diversi e ben distinguibili insiemi o cluster. Questa esperienza insegna che l'informazione è contenuta a diversi livelli in un testo. Non solo è del tutto plausibile ipotizzare che le unità ancora più elementari di informazione rispetto alle singole lettere che sono i fonemi contengano ancora più informazione. Si immagini di poter memorizzare un testo parlato nei singoli foneni di una persona. Sicuramente si tratterebbe di un maggiore patrimonio informativo delle singole lettere che se vogliamo sono una discretizzazione, un appiattimento su un singolo simbolo, di suoni simili ma diversi. Ma come detto anche le sole singole lettere contengono informazione sufficiente a distinguere ad esempio gli autori. Del resto è evidente che si potrebbero distinguere dei parlati per le diverse inflessioni della pronuncia. Nel prossimo esempio, invece, l'oggetto della nostre attenzioni come unità minima informativa sarà la singola parola.

COMPARAZIONE TRA DUE TESTI

Un problema ricorrente per chi fa analisi quantitative su testi presenti ad esempio sul web è mettere in relazione diversi scritti come dei singoli post per i forum. Verificare se due post abbiano elementi in comune e che tipo di relazione possa esserci tra essi. Un modo potrebbe essere applicare anche in questo caso tecniche di clustering per raggruppare rispetto a qualche parametro più testi. Ma tale metodo nel caso specifico è da scartare per due ragioni: la non significatività dei testi che spesso sono troppo brevi e l'impossibilità di avere numerosi testi tali da poter

sottoporre ad analisi. Si può quindi pensare di sviluppare un procedimento che possa confrontare semplicemente due testi sulla base della semplice informazione derivante dalla frequenza delle parole. Indicando così un indice di relazione o meglio di correlazione tra i due testi in questione. Ovviamente, tale valore dipenderà dalla presenza di parole comuni nei due testi. Saranno definiti due indici di relazione tra il primo ed il secondo testo e tra il secondo ed il primo che possono anche divergere. Se, infatti, il primo testo è più breve e contiene molte parole presenti anche sul secondo presenterà una forte relazione con esso. Viceversa, il secondo avrà un grado di relazione minore rispetto al primo testo. L'indice sarà normalizzato in modo che possa variare tra zero e uno, con l'intendimento che valori vicini allo zero indichino deboli relazioni mentre valori vicino all'uno forti relazioni. È utile sottolineare che si tratta di indici che sono stati prodotti sulla base di criteri statistici ma che comunque usano procedimenti euristici e che quindi possono essere migliorati e vanno opportunamente interpretati. Ad esempio è impensabile avere relazioni con indici vicino all'uno, si verificano in rari casi. Già valori dell'ordine del decimo (0,1) possono indicare gradi "significativi" di relazione. Nello sviluppo del metodo sono state opportunamente considerate tutte le parole presenti nel testo, anche elementi che potrebbero sembrare privi di informazioni come le proposizioni e gli avverbi. Infatti, la ripetizioni oltre un limite fisiologico di tali parole può essere un elemento che contraddistingue un testo. Inoltre, si è tenuto conto per ogni parola di un determinato peso. Si pensi a parole inconsuete che si vogliono evidenziare o marcare, come termini usati in ambiti particolari come quelli sessuali o estremisti. Dando ad esse peso maggiore determineranno maggiore rilevanza nell'indice di relazione. Ovviamente, si tratterebbe di definire manualmente i pesi delle singole parole nel testo. Ma tranne in rari casi tale procedimento non è consigliabile essendo tedioso. Allora ho pensato di produrre in automatico tali pesi utilizzando un dizionario "universale". Non ho nozione se ne esista già un lavoro simile, ma penso proprio di si. Si tratta di un dizionario, nel nostro caso della lingua italiana, costruito analizzando una enorme quantità di testi il più eterogenei possibili (provenienti da più ambiti come il letterario, lo scientifico, lo sportivo, il giornalistico e quant altro) che contiene per ogni parola la sua frequenza relativa, ossia la frequenza fratto il numero totale delle parole analizzate; il che indica se la parola è comune o rara. Quindi si potrebbe pensare di far pesare maggiormente le parole rare. Questo perché due testi sono maggiormente in relazione se contengono entrambi una parola desueta piuttosto che una parola di uso comune. Il peso potrebbe quindi essere inversamente proporzionale alla frequenza relativa. Detto ciò l'indice di relazione del testo 1 verso il testo 2 può essere espresso dalla seguente formula.

Si calcola come il rapporto tra la sommatoria delle frequenze relative per i rispettivi pesi rispetto alle parole comuni ai due testi e la stessa sommatoria rispetto a tutte le parole del testo. Se per assurdo tutte le parole presenti nel testo 1 sono anche presenti nel testo 2 allora i due termini della frazione coincidono e l'indice risulta massimo, ovvero 1, in tal caso la relazione del primo verso il secondo testo è massima. Se, invece, nessuna parola del primo testo coincide con il secondo allora in numeratore è zero così come il rapporto, la relazione tra i due testi è inesistente.

IMPLEMENTAZIONE

Nel produrre l'implementazione darò le indicazioni per sviluppare l'algoritmo ed un adeguata struttura dati e presenterò un esempio sviluppato su foglio di calcolo per il calcolo dell'indice di relazione. I due testi siano presenti su due file di nome *testo1.txt* e *testo2.txt*, qualora non lo fossero una routine a parte potrebbe caricarli ad esempio dai post di interesse nei suddetti due file. I passi dell'algoritmo possono essere espressi come segue:

```
impronta1=caricatesto("testo1.txt")
impronta2=caricatesto("testo2.txt")
definiscipesi(dizionario, impronta1)
definiscipesi(dizionario, impronta2)
```

LEGGI QUANTITATIVE DELLA LINGUA

La sinonimia, presente in molte lingue, se non tutte, è considerata da molti un elemento di ricchezza della lingua stessa. Infatti, scrittori usano le molteplici possibilità nella scelta di parole per definire un proprio carattere stilistico. Ad ogni modo vi sono parole più usate di altre. Alcuni studi hanno mostrato che le parole più antiche di origine latina e greca sono le più usate ma anche le più soggette a modificazioni ad esempio fonetiche. Ma la cosa sorprendente è che esistono delle leggi quantitative che valgono per molte lingue, per alcune di più che per altre. Ad esempio se si ordinano in un testo molto grande le parole per la loro frequenza in modo non crescente (ossia dalla più frequente alla meno frequente) il numero d'ordine in questa lista sarà il rango. Bene, il prodotto tra il rango e la frequenza al quadrato di tutte le parole tende ad essere costante. K=r*f*f. Ma esistono molte altre relazioni matematiche a cui ubbidiscono le produzioni linguistiche. Ad esempio l'entropia è definita anche in questo ambito come grado di indeterminazione ed è calcolabile mediante delle formule. La quantità di informazione è tanto più grande quanto maggiore il grado di indeterminazione eliminato da una certa informazione. Questo segue una logica che si riconduce alla teoria di Shannon.

```
r12 = compara(impronta1, impronta2)
r21 = compara(impronta2, impronta1)
```

caricatesto è molto importante poiché deve farsi carico di segmentare il testo in unità informative, ossia le parole, tenendo conto dei terminali che le dividono, che oltre allo spazio sono tutti i caratteri di punteggiatura e speciali. Per ogni parola estratta si deve verificare che non sia già presente nel vettore che si sta caricando. Se è presente si deve semplicemente incrementare il campo relativo alla frequenza assoluta (si veda la dichiarazione in C++ proposta più avanti), altrimenti si deve inserire in modo ordinato nel vettore il nuovo valore settando a 1 il valore corrispondente alla frequenza assoluta. Terminato il ciclo bisogna innescarne un altro per il calcolo delle frequenze relative calcolate come frequenza assoluta fratto numero totale di parole. Si producono così i vettori di record indicati con *impronta1* e *impronta2*. Questi sono aggiornati con i pesi relativi alle singole parole attingendo ad un dizionario che abbia le caratteristiche descritte precedentemente. Infine, è possibile calcolare i due indici comparando le due impronte e con il procedimento descritto.
Una plausibile struttura dati C++ è la seguente:

```
const int lmax=10000;
const int maxpar=15;

struct occorrenza
{ char *parola;
  int freq, peso;
  float freqr;
};
occorrenza impronta1[lmax],impronta2[lmax];
int n1,n2;
```

Con *n1* e *n2* reali cardinalità dei due vettori impronta. La struttura singola prende il nome di *occorrenza*. In Excel è stato simulato un caso reale, ma molto semplicistico; si tratta di testi molto brevi che nella realtà si dovrebbero evitare poiché mancano di significatività. Nel caso specifico vengono usati perché con un basso numero di parole si può sviluppare più agevolmente e con più chiarezza l'esempio.

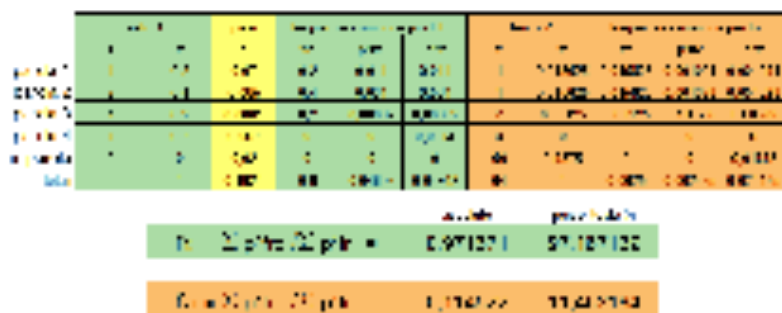



Fig. 1: I due indici di relazione e valori intermedi per effettuare il calcolo

L'intera tabella che porta al calcolo dei due indici è una cattura del foglio excel ed è riportata in figura 1. In verde sono riportati i dati relativi al testo 1, mentre in arancio quelli relativi al testo 2, in giallo i pesi. Vi sono in esame 64 parole di cui le prime quattro sono presenti nel testo 1 con le frequenze descritte. Le altre 60 non sono presenti nel testo 1, mentre si ritrovano nel testo 2. Delle prime quattro parole le prime tre sono anche presenti nel testo 2. Le due colonne dei dati delle frequenze si possono ottenere con un apposito programma, ad esempio in C++; e poi importate in excel. Il foglio di calcolo ha il vantaggio di essere facilmente manipolato, in particolar modo nella fase di costruzione di una giusta formula che esprima la correlazione. Si suppone che i pesi siano individuati con un dizionario "universale" o opportunamente modificati da utenti autorizzati. Le frequenze relative si calcolano come rapporto fra le frequenze e il totale delle parole del testo di riferimento. Le frequenze relative comuni frc presentano gli stessi valori delle frequenze relative ma si valutano nel solo caso siano presenti anche nel secondo testo. Il prodotto di tale colonna rispetto al peso è calcolato nella colonna adiacente. Rispetto a questa serie è importante la somma. I due indici si calcolano con la formula prima descritta. Con i dati presenti si ottiene un risultato degno di attenzione. Il primo indice è alto circa 97% mentre il secondo è di circa 11% che è molto meno ma non va considerato basso. È comunque certo che il primo testo ha un elevato grado di attinenza con il secondo avendo molte parole presenti in esso. Mentre il secondo ha solo una bassa percentuale di termini presenti nell'altro testo cosicché si stabilisce una minore relazione. È interessante fare piccoli esperimenti. Se ad esempio si fa in modo che il testo più piccolo presenti tutte le sue parole nel testo più grande, in tal caso bisogna portare ad un valore maggiore di zero la frequenza assoluta della quarta parola, si vede come il primo indice passa a 1, ossia relazione massima. Se non si hanno parole comuni si registra zero. E così si può sperimentare e verificare come variano di due indici.

CONCLUSIONI

L'ambito che abbiamo appena sfiorato in questo numero è davvero vasto. Ciò sarà emerso dalla difficoltà palpabile di fare una sintesi della teoria ad oggi prodotta. È comunque appassionante sviluppare esperimenti e verificare l'enorme mole informativa presente in molte forme della lingua. Vi aspetto per l'esplorazione di ulteriori aspetti legati a tali questioni.

Fabio Grimaldi